H

Hamilton Cycles in Random Intersection Graphs

Charilaos Efthymiou[1] and Paul (Pavlos) Spirakis[2,3,4]
[1]Department of Computer Engineering and Informatics, University of Patras, Patras, Greece
[2]Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
[3]Computer Science, University of Liverpool, Liverpool, UK
[4]Computer Technology Institute (CTI), Patras, Greece

Keywords

Stochastic order relations between Erdös–Rényi random graph model and random intersection graphs; Threshold for appearance of Hamilton cycles in random intersection graphs

Years and Authors of Summarized Original Work

2005; Efthymiou, Spirakis

Problem Definition

E. Marczewski proved that every graph can be represented by a list of sets where each vertex corresponds to a set and the edges to nonempty intersections of sets. It is natural to ask what sort of graphs would be most likely to arise if the list of sets is generated randomly.

Consider the model of random graphs where each vertex chooses randomly from a universal set the members of its corresponding set, each independently of the others. The probability space that is created is the space of random intersection graphs, $G_{n,m,p}$, where n is the number of vertices, m is the cardinality of a universal set of elements and p is the probability for each vertex to choose an element of the universal set. The model of random intersection graphs was first introduced by M. Karoński, E. Scheinerman, and K. Singer-Cohen in [4]. A rigorous definition of the model of random intersection graphs follows:

Definition 1 Let n, m be positive integers and $0 \leq p \leq 1$. The random intersection graph $G_{n,m,p}$ is a probability space over the set of graphs on the vertex set $\{1, \ldots, n\}$ where each vertex is assigned a random subset from a fixed set of m elements. An edge arises between two vertices when their sets have at least a common element. Each random subset assigned to a vertex is determined by

$$\mathbf{Pr}\left[\text{vertex } i \text{ chooses element } j\right] = p$$

with these events mutually independent.

A common question for a graph is whether it has a cycle, a set of edges that form a path so that the

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

first and the last vertex is the same, that visits *all* the vertices of the graph exactly once. We call this kind of cycle the *Hamilton cycle* and the graph that contains such a cycle is called a *Hamiltonian graph*.

Definition 2 Consider an undirected graph $G = (V, E)$ where V is the set of vertices and E the set of edges. This graph contains a Hamilton cycle if and only if there is a simple cycle that contains each vertex in V.

Consider an instance of $G_{n,m,p}$, for specific values of its parameters n, m, and p, what is the probability of that instance to be Hamiltonian? Taking the parameter p, of the model, to be a function of n and m, in [2], a threshold function $P(n,m)$ has been found for the graph property "Contains a Hamilton cycle"; i.e., a function $P(n,m)$ is derived such that

if $p(n,m) \ll P(n,m)$

$$\lim_{n,m\to\infty} \mathbf{Pr}\left[G_{n,m,p}\text{Contains Hamilton cycle}\right] = 0$$

if $p(n,m) \gg P(n,m)$

$$\lim_{n,m\to\infty} \mathbf{Pr}\left[G_{n,m,p}\text{Contains Hamilton cycle}\right] = 1$$

When a graph property, such as "Contains a Hamilton cycle," holds with probability that tends to 1 (or 0) as n, m tend to infinity, then it is said that this property holds (does not hold), "almost surely" or "almost certainly."

If in $G_{n,m,p}$ the parameter m is very small compared to n, the model is not particularly interesting and when m is exceedingly large (compared to n) the behavior of $G_{n,m,p}$ is essentially the same as the Erdös–Rényi model of random graphs (see [3]). If someone takes $m = \lceil n^{\alpha} \rceil$, for fixed real $\alpha > 0$, then there is some deviation from the standard models, while allowing for a natural progression from sparse to dense graphs. Thus, the parameter m is assumed to be of the form $m = \lceil n^{\alpha} \rceil$ for some fixed positive real α.

The proof of existence of a Hamilton cycle in $G_{n,m,p}$ is mainly based on the establishment of a *stochastic order relation* between the model $G_{n,m,p}$ and the Erdös–Rényi random graph model $G_{n,\hat{p}}$.

Definition 3 Let n be a positive integer, $0 \le \hat{p} \le 1$. The random graph $G(n, \hat{p})$ is a probability space over the set of graphs on the vertex set $\{1, \ldots, n\}$ determined by

$$\mathbf{Pr}\,[i, j] = \hat{p}$$

with these events mutually independent.

The stochastic order relation between the two models of random graphs is established in the sense that if $\mathcal{A}$ is an increasing graph property, then it holds that

$$\mathbf{Pr}\left[G_{n,\hat{p}} \in \mathcal{A}\right] \le \mathbf{Pr}\left[G_{n,m,p} \in \mathcal{A}\right]$$

where $\hat{p} = f(p)$. A graph property $\mathcal{A}$ is increasing if and only if given that $\mathcal{A}$ holds for a graph $G(V, E)$ then $\mathcal{A}$ holds for any $G(V, E')$: $E' \supseteq E$.

Key Results

Theorem 1 *Let $m = \lceil n^{\alpha} \rceil$, where α is a fixed real positive, and C_1, C_2 be sufficiently large constants. If*

$$p \ge C_1 \frac{\log n}{m} \quad \text{for} \quad 0 < \alpha < 1 \quad \text{or}$$

$$p \ge C_2 \sqrt{\frac{\log n}{nm}} \quad \text{for} \quad \alpha > 1$$

then almost all $G_{n,m,p}$ are Hamiltonian. Our bounds are asymptotically tight.

Note that the theorem above says nothing when $m = n$, i.e., $\alpha = 1$.

Applications

The Erdös–Rényi model of random graphs, $G_{n,p}$, is exhaustively studied in computer science because it provides a framework for studying practical problems such as "reliable network computing" or it provides a "typical instance" of a graph and thus it is used for average case analysis of graph algorithms. However, the simplicity of $G_{n,p}$ means it is not able to capture satisfactorily many practical problems in computer science. Basically, this is because of the fact that in many problems independent edge-events are not well justified. For example, consider a graph whose vertices represent a set of objects that either are placed or move in a specific geographical region, and the edges are radio communication links. In such a graph, we expect that, any two vertices u, w are more likely to be adjacent to each other, than any other, arbitrary, pair of vertices, if both are adjecent to a third vertex v. Even epidemiological phenomena (like the spread of disease) tend to be more accurately captured by this proximity-sensitive random intersection graph model. Other applications may include oblivious resource sharing in a distributive setting, interaction of mobile agents traversing the web etc.

The model of random intersection graphs $G_{n,m,p}$ was first introduced by M. Karońsky, E. Scheinerman, and K. Singer-Cohen in [4] where they explored the evolution of random intersection graphs by studying the thresholds for the appearance and disappearance of small induced subgraphs. Also, J.A. Fill, E.R. Scheinerman, andK. Singer Cohen in [3] proved an equivalence theorem relating the evolution of $G_{n,m,p}$ and $G_{n,p}$, in particular they proved that when $m = n^{\alpha}$ where $\alpha > 6$, the total variation distance between the graph random variables has limit 0. S. Nikoletseas, C. Raptopoulos, and P. Spirakis in [8] studied the existence and the efficient algorithmic construction of close to optimal independent sets in random intersection graphs. D. Stark in [11] studied the degree of the vertices of the random intersection graphs. However, after [2], Spirakis and Raptopoulos, in [10], provide algorithms that construct Hamilton cycles in instances of $G_{n,m,p}$, for p above the Hamiltonicity threshold. Finally, Nikoletseas et al. in [7] study the mixing time and cover time as the parameter p of the model varies.

Open Problems

As in many other random structures, e.g., $G_{n,p}$ and random formulae, properties of random intersection graphs also appear to have threshold behavior. So far threshold behavior has been studied for the induced subgraph appearance and hamiltonicity.

Other fields of research for random intersection graphs may include the study of connectivity behavior, of the model i.e., the path formation, the formation of giant components. Additionally, a very interesting research question is how cover and mixing times vary with the parameter p, of the model.

Cross-References

▶ Independent Sets in Random Intersection Graphs

Recommended Reading

1. Alon N, Spencer JH (2000) The probabilistic method, 2nd edn. Wiley, New York
2. Efthymiou C, Spirakis PG (2005) On the existence of Hamilton cycles in random intersection graphs. In: Proceedings of the 32nd ICALP. LNCS, vol 3580. Springer, Berlin/Heidelberg, pp 690–701
3. Fill JA, Scheinerman ER, Singer-Cohen KB (2000) Random intersection graphs when m = ω(n): an equivalence theorem relating the evolution of the G(n, m, p) and G(n, p) models. Random Struct Algorithms 16:156–176
4. Karoński M, Scheinerman ER, Singer-Cohen K (1999) On random intersection graphs: the subgraph problem. Comb Probab Comput 8:131–159
5. Komlós J, Szemerédi E (1983) Limit distributions for the existence of Hamilton cycles in a random graph. Discret Math 43:55–63
6. Korshunov AD (1977) Solution of a problem of P. Erdös and A. Rényi on Hamilton cycles in non-oriented graphs. Metody Diskr Anal Teoriy Upr Syst Sb Trubov Novosibrirsk 31:17–56
7. Nikoletseas S, Raptopoulos C, Spirakis P (2007) Expander properties and the cover time of random intersection graphs. In: Proceedings of the 32nd MFCS. Springer, Berlin/Heidelberg, pp 44–55
8. Nikoletseas S, Raptopoulos C, Spirakis P (2004) The existence and efficient construction of large independent sets in general random intersection graphs.

In: Proceedings of the 31st ICALP. LNCS, vol 3142. Springer, Berlin/Heidelberg, pp 1029–1040
9. Singer K (1995) Random intersection graphs. PhD thesis, The Johns Hopkins University, Baltimore
10. Spirakis PG, Raptopoulos C (2005) Simple and efficient greedy algorithms for Hamilton cycles in random intersection graphs. In: Proceedings of the 16th ISAAC. LNCS, vol 3827. Springer, Berlin/Heidelberg, pp 493–504
11. Stark D (2004) The vertex degree distribution of random intersection graphs. Random Struct Algorithms 24:249–258

Haplotype Inference on Pedigrees Without Recombinations

Mee Yee Chan[1], Wun-Tat Chan[2], Francis Y.L. Chin[1], Stanley P.Y. Fung[3], and Ming-Yang Kao[4]
[1]Department of Computer Science, University of Hong Kong, Hong Kong, China
[2]College of International Education, Hong Kong Baptist University, Hong Kong, China
[3]Department of Computer Science, University of Leicester, Leicester, UK
[4]Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Keywords

Computational biology; Haplotype inference; Pedigree; Recombination

Years aud Authors of Summarized Original Work

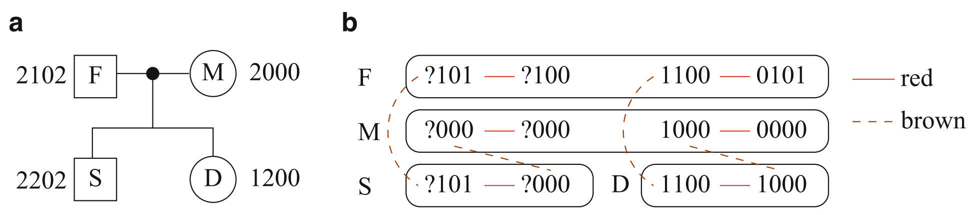
2009; Chan, Chan, Chin, Fung, Kao

Problem Definition

In many diploid organisms like humans, chromosomes come in pairs. Genetic variation occurs in some "positions" along the chromosomes. These genetic variations are commonly modelled in the form of *single nucleotide polymorphisms* (SNPs) [5], which are the nucleotide sites where more than one nucleotide can occur. A *haplotype* is the sequence of linked SNP genetic markers (small segments of DNA) on a single chromosome. However, experiments often yield *genotypes*, which is a blend of the two haplotypes of the chromosome pair. It is more useful to have information on the haplotypes, thus giving rise to the computational problem of inferring haplotypes from genotypes.

The physical position of a marker on a chromosome is called a *locus* and its state is called an *allele*. SNP are often *biallelic*, i.e., the allele can take on two different states, corresponding to two different nucleotides. In the language of computer science, the allele of a biallelic SNP can be denoted by 0 and 1, and a haplotype with m loci is represented as a length-m string in $\{0,1\}^m$ and a genotype as a length-m string in $\{0,1,2\}^m$. Consider a haplotype pair $\langle h_1, h_2\rangle$ and a corresponding genotype g. For each locus, if both haplotypes show a 0, then the genotype must also be 0, and if both haplotypes show a 1, the genotype must also be 1. These loci are called *homozygous*. If however one of the haplotypes shows a 0 and the other a 1, the genotype shows a 2 and the locus is called *heterozygous*. This is called *SNP consistency*. For example, considering a single individual, the genotype $g = 012212$ has four SNP-consistent haplotype pairs: $\{\langle 01\underline{11}1\underline{1}, 01\underline{00}1\underline{0}\rangle, \langle 01\underline{11}1\underline{0}, 01\underline{00}1\underline{1}\rangle, \langle 01\underline{10}1\underline{1}, 01\underline{01}1\underline{0}\rangle, \langle 01\underline{10}1\underline{0}, 01\underline{01}1\underline{1}\rangle\}$. In general, if a genotype has s heterozygous loci, it can have 2^{s-1} SNP-consistent haplotype solutions.

Haplotypes are passed down from an individual to its descendants. *Mendelian consistency* requires that, in the absence of *recombinations* or mutations, each child inherits one haplotype from one of the two haplotypes of the father and inherits the other haplotype from the mother similarly. This gives us more information to infer haplotypes when we are given a *pedigree*. The computational problem is therefore, given a pedigree with n individuals where each individual is associated with a genotype of length m, find an assignment of a pair of haplotypes to each individual such that SNP consistency



Haplotype Inference on Pedigrees Without Recombinations, Fig. 1 (**a**) Example of a pedigree with four nodes. (**b**) The graph G with 12 vertices, 6 *red edges*, and 4 *brown edges*. Each vector is a vertex in G. Vector pairs enclosed by *rounded rectangles* belong to the same individual

and Mendelian consistency are obeyed for each individual. In rare cases (especially for humans) [3], the pedigree may contain *mating loops*: a mating loop is formed when, for example, there is a marriage between descendants of a common ancestor.

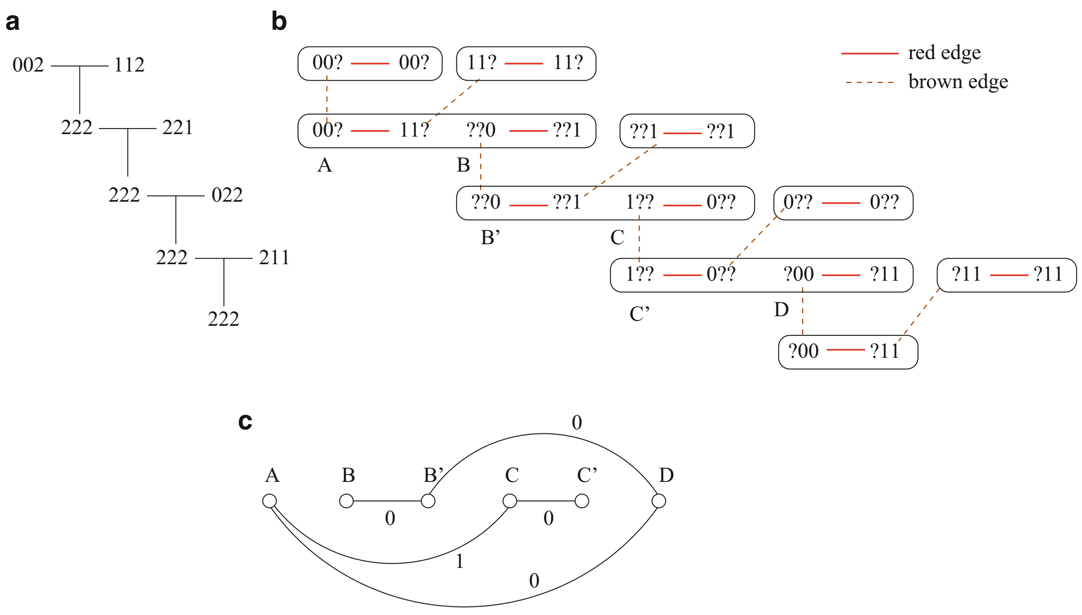
As a simple example, consider the pedigree in Fig. 1a for a family of four individuals and their genotypes. Due to SNP consistency, mother M's haplotypes must be $\langle 0000, 1000 \rangle$ (the order does not matter). Similarly, daughter D's haplotypes must be $\langle 1000, 1100 \rangle$. Now we apply Mendelian consistency to deduce that D must obtain the 1000 haplotype from M since neither of father F's haplotypes can be 1000 (considering locus 2). Therefore, D obtains 1100 from F, and F's haplotypes must be $\langle 0101, 1100 \rangle$. With F's and M's haplotypes known, the only solution for the haplotypes of son S that is consistent with his genotype 2202 is $\langle 0101, 1000 \rangle$.

Key Results

While this kind of deduction might appear to be enough to resolve all haplotype values, it is not the case. As we will shortly see, there are "long-distance" constraints that need to be considered. These constraints can be represented by a system of linear equations in GF(2) and solved using Gaussian elimination. This gives a $O(m^3 n^3)$ time algorithm [3]. Subsequent papers try to capture or solve the constraints more economically. The time complexity was improved in [6] to $O(mn^2 + n^3 \log^2 n \log \log n)$ by eliminating redundant equations and using low-stretch spanning trees. A different approach was used in [1], representing the constraints by the parity of edge labels of some auxiliary graphs and finding solutions of these constraints using graph traversal without (directly) solving a system of linear equations. This gives a linear $O(mn)$ time algorithm, although it only works for the case with no mating loops and only produces one particular solution even when the pedigree admits more than one solution. Later algorithms include [4] which returns the full set of solutions in optimal time (again without mating loops) and [2] which can handle mating loops and runs in $O(kmn + k^2 m)$ time where k is the number of mating loops.

In the following we sketch the idea behind the linear time algorithm in [1]. Each individual only has a pair of haplotypes, but the algorithm first produces a number of vector pairs for each individual, one vector pair for each trio (a father-mother-child triplet) that this individual belongs to. Each vector pair represents the information about the two haplotypes of this individual that can be derived by considering this trio only. These vector pairs will eventually be "unified" to become a single pair.

For the pedigree in Fig. 1a, the algorithm first produces the graph G in Fig. 1b, which has two connected components for the two trios F-M-S and F-M-D. The rule for enforcing SNP consistency (Mendelian consistency) is that the unresolved loci values, i.e., the ? values, must be different (same) at opposite ends of a red (brown) edge. There is only one way to unify the vector pairs of F consistently (due to locus 4): ?101 must correspond to 0101. We add an edge between these two vectors to represent the fact that they should be identical. Then all ? values



Haplotype Inference on Pedigrees Without Recombinations, Fig. 2 An example showing how constraints are represented by labeled edges in another graph. (**a**) The pedigree. (**b**) The local graph G. (**c**) The parity constraint graph J. Three constraints are added

can be resolved by traversing the now-connected graph and applying the aforementioned rules for enforcing consistency.

However, consider another pedigree in Fig. 2a. The previous steps can only produce Fig. 2b, which has four connected components and unresolved loci. We need to decide for A and B whether $A = 00?$ should connect to B =??0 or its complement ??1 and similarly for B' and C, etc. Observe that a path between A and C must go through an odd number of red edges since locus 1 changes from 0 to 1. To capture this type of long-distance constraints, we construct a *parity constraint graph* J where the edge labels represent the parity constraints; see Fig. 2c. In effect, J represents a set of linear equations in GF(2); in Fig. 2c, the equations are $x_{AB}+x_{B'C} = 1, x_{B'C}+x_{C'D} = 0$, and $x_{AB}+x_{B'C}+x_{C'D} = 0$.

Finally, we can traverse J along the unique path between any two nodes; the parity of this path tells us how to merge the vector pairs in G. For example, the parity between A and B should be 0, indicating 00? in A should connect to ??0 in B (so both become 000), while the parity between B' and C is 1, so B' and C should be 000 and 111, respectively.

Cross-References

▸ Beyond Evolutionary Trees
▸ Musite: Tool for Predicting Protein Phosphorylation Sites
▸ Sequence and Spatial Motif Discovery in Short Sequence Fragments

Recommended Reading

1. Chan MY, Chan WT, Chin FYL, Fung SPY, Kao MY (2009) Linear-time haplotype inference on pedigrees without recombinations and mating loops. SIAM J Comput 38(6):2179–2197
2. Lai EY, Wang WB, Jiang T, Wu KP (2012) A linear-time algorithm for reconstructing zero-recombinant haplotype configuration on a pedigree. BMC Bioinformatics 13(S-17):S19
3. Li J, Jiang T (2003) Efficient inference of haplotypes from genotypes on a pedigree. J. Bioinformatics Comput Biol 1(1):41–69
4. Liu L and Jiang T (2010) A linear-time algorithm for reconstructing zero-recombinant haplotype configuration on pedigrees without mating loops. J Combin Optim 19:217–240
5. Russo E, Smaglik P (1999) Single nucleotide polymorphism: big pharmacy hedges its bets. The Scientist, 13(15):1, July 19, 1999

6. Xiao J, Liu L, Xia L, Jiang T (2009) Efficient algorithms for reconstructing zero-recombinant haplotypes on a pedigree based on fast elimination of redundant linear equations. SIAM J Comput 38(6):2198–2219

Hardness of Proper Learning

Vitaly Feldman
IBM Research – Almaden, San Jose, CA, USA

Keywords

DNF; function representation; NP-hardness of learning; PAC learning; Proper learning; representation-based hardness

Years and Authors of Summarized Original Work

1988; Pitt, Valiant

Problem Definition

The work of Pitt and Valiant [18] deals with learning Boolean functions in the Probably Approximately Correct (PAC) learning model introduced by Valiant [19]. A learning algorithm in Valiant's original model is given random examples of a function $f : \{0,1\}^n \to \{0,1\}$ from a representation class $\mathcal{F}$ and produces a hypothesis $h \in \mathcal{F}$ that closely approximates f. Here, a *representation class* is a set of functions and a language for describing the functions in the set. The authors give examples of natural representation classes that are NP-hard to learn in this model, whereas they can be learned if the learning algorithm is allowed to produce hypotheses from a richer representation class $\mathcal{H}$. Such an algorithm is said to learn $\mathcal{F}$ by $\mathcal{H}$; learning $\mathcal{F}$ by $\mathcal{F}$ is called *proper* learning.

The results of Pitt and Valiant were the first to demonstrate that the choice of representation of hypotheses can have a dramatic impact on the computational complexity of a learning problem. Their specific reductions from NP-hard problems are the basis of several other follow-up works on the hardness of proper learning [1, 3, 7].

Notation

Learning in the PAC model is based on the assumption that the unknown function (or *concept*) belongs to a certain class of concepts $\mathcal{C}$. In order to discuss algorithms that learn and output functions, one needs to define how these functions are represented. Informally, a representation for a concept class $\mathcal{C}$ is a way to describe concepts from $\mathcal{C}$ that defines a procedure to evaluate a concept in $\mathcal{C}$ on any input. For example, one can represent a conjunction of input variables by listing the variables in the conjunction. More formally, a representation class can be defined as follows.

Definition 1 A *representation class* $\mathcal{F}$ is a pair $(L, \mathcal{R})$ where

- L is a language over some fixed finite alphabet (e.g., $\{0,1\}$);
- $\mathcal{R}$ is an algorithm that for $\sigma \in L$, on input $(\sigma, 1^n)$ returns a Boolean circuit over $\{0,1\}^n$.

In the context of efficient learning, only efficient representations are considered, or, representations for which $\mathcal{R}$ is a polynomial-time algorithm. The concept class represented by $\mathcal{F}$ is the set of functions over $\{0,1\}^n$ defined by the circuits in $\{\mathcal{R}(\sigma, 1^n) \mid \sigma \in L\}$. For a Boolean function f, "$f \in \mathcal{F}$" means that f belongs to the concept class represented by $\mathcal{F}$ and that there is a $\sigma \in L$ whose associated Boolean circuit computes f. For most of the representations discussed in the context of learning, it is straightforward to construct a language L and the corresponding translating function $\mathcal{R}$, and therefore, they are not specified explicitly.

Associated with each representation is the complexity of describing a Boolean function using this representation. More formally, for a Boolean function $f \in \mathcal{C}$, $\mathcal{F}\text{-}\mathtt{size}(f)$ is the length of the shortest way to represent f using $\mathcal{F}$, or $\min\{|\sigma| \mid \sigma \in L,\ \mathcal{R}(\sigma, 1^n) \equiv f\}$.

We consider Valiant's PAC model of learning [19], as generalized by Pitt and Valiant [18].

In this model, for a function f and a distribution $\mathcal{D}$ over X, an *example oracle* $\mathrm{EX}(f, \mathcal{D})$ is an oracle that, when invoked, returns an example $\langle x, f(x)\rangle$, where x is chosen randomly with respect to $\mathcal{D}$, independently of any previous examples. For $\epsilon \geq 0$, we say that function g ϵ-approximates a function f with respect to distribution $\mathcal{D}$ if $\mathbf{Pr}_{\mathcal{D}}[f(x) \neq g(x)] \leq \epsilon$.

Definition 2 A representation class $\mathcal{F}$ is *PAC learnable* by representation class $\mathcal{H}$ if there exists an algorithm that for every $\epsilon > 0$, $\delta > 0$, n, $f \in \mathcal{F}$, and distribution $\mathcal{D}$ over X, given ϵ, δ, and access to $\mathrm{EX}(f, \mathcal{D})$, runs in time polynomial in $n, s = \mathcal{F}\text{-}\mathtt{size}(c)$, $1/\epsilon$ and $1/\delta$, and outputs, with probability at least $1-\delta$, a hypothesis $h \in \mathcal{H}$ that ϵ-approximates f.

A DNF expression is defined as an OR of ANDs of literals, where a *literal* is a possibly negated input variable. We refer to the ANDs of a DNF formula as its *terms*. Let $\mathsf{DNF}(k)$ denote the representation class of k-term DNF expressions. Similarly, a CNF expression is an OR of ANDs of literals. Let k-CNF denote the representation class of CNF expressions with each AND having at most k literals.

For a real-valued vector $c \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, a *linear threshold function* (also called a *halfspace*) $T_{c,\theta}(x)$ is the function that equals 1 if and only if $\sum_{i \leq n} c_i x_i \geq \theta$. The representation class of Boolean threshold functions consists of all linear threshold functions with $c \in \{0, 1\}^n$ and θ an integer.

Key Results

Theorem 1 ([18]) *For every $k \geq 2$, the representation class of* $\mathsf{DNF}(k)$ *is not properly learnable unless* $\mathsf{RP} = \mathsf{NP}$.

More specifically, Pitt and Valiant show that learning $\mathsf{DNF}(k)$ by $\mathsf{DNF}(\ell)$ is at least as hard as coloring a k-colorable graph using ℓ colors. For the case $k = 2$, they obtain the result by reducing from Set Splitting (see [9] for details on the problems). Theorem 1 is in sharp contrast with the fact that $\mathsf{DNF}(k)$ is learnable by k-CNF [19].

Theorem 2 ([18]) *The representation class of Boolean threshold functions is not properly learnable unless* $\mathsf{RP} = \mathsf{NP}$.

This result is obtained via a reduction from the NP-complete Zero-One Integer Programming problem (see [9] (p.245) for details on the problem). The result is contrasted by the fact that general linear thresholds are properly learnable [4].

These results show that using a specific representation of hypotheses forces the learning algorithm to solve a combinatorial problem that can be NP-hard. In most machine learning applications it is not important which representation of hypotheses is used as long as the value of the unknown function is predicted correctly. Therefore, learning in the PAC model is now defined without any restrictions on the output hypothesis (other than it being efficiently evaluatable). Hardness results in this setting are usually based on cryptographic assumptions (cf. [15]).

Hardness results for proper learning based on assumption $\mathsf{NP} \neq \mathsf{RP}$ are now known for several other representation classes and for other variants and extensions of the PAC learning model. Blum and Rivest show that for any $k \geq 3$, unions of k halfspaces are not properly learnable [3]. Hancock et al. prove that decision trees (cf. [16] for the definition of this representation) are not learnable by decision trees of somewhat larger size [11]. This result was strengthened by Alekhnovich et al. who also proved that intersections of two halfspaces are not learnable by intersections of k halfspaces for any constant k, general DNF expressions are not learnable by unions of halfspaces (and in particular are not properly learnable) and k-*juntas* are not properly learnable [1]. Further, DNF expressions remain NP-hard to learn properly even if *membership queries*, or the ability to query the unknown function at any point, are allowed [7]. Khot and Saket show that the problem of learning intersections of two halfspaces remains NP-hard even if a hypothesis with any constant error smaller than $1/2$ is required [17]. No efficient algorithms or hardness results are known for any of the above learning problems if no restriction is placed on the representation of hypotheses.

The choice of representation is important even in powerful learning models. Feldman proved that n^c-term DNF are not properly learnable for any constant c even when the distribution of examples is assumed to be uniform and membership queries are available [7]. This contrasts with Jackson's celebrated algorithm for learning DNF in this setting [13], which is not proper.

In the *agnostic learning* model of Haussler [12] and Kearns et al. [14], even the representation classes of conjunctions, decision lists, halfspaces, and parity functions are NP-hard to learn properly (cf. [2, 6, 8, 10] and references therein). Here again the status of these problems in the representation-independent setting is largely unknown.

Applications

A large number of practical algorithms use representations for which hardness results are known (most notably decision trees, halfspaces, and neural networks). Hardness of learning $\mathcal{F}$ by $\mathcal{H}$ implies that an algorithm that uses $\mathcal{H}$ to represent its hypotheses will not be able to learn $\mathcal{F}$ in the PAC sense. Therefore such hardness results elucidate the limitations of algorithms used in practice. In particular, the reduction from an NP-hard problem used to prove the hardness of learning $\mathcal{F}$ by $\mathcal{H}$ can be used to generate hard instances of the learning problem.

Open Problems

A number of problems related to proper learning in the PAC model and its extensions are open. Almost all hardness of proper learning results are for learning with respect to unrestricted distributions. For most of the problems mentioned in section "Key Results" it is unknown whether the result is true if the distribution is restricted to belong to some natural class of distributions (e.g., product distributions). It is unknown whether decision trees are learnable properly in the PAC model or in the PAC model with membership queries. This question is open even in the PAC model restricted to the uniform distribution only. Note that decision trees are learnable (non-properly) if membership queries are available [5] and are learnable properly in time $O(n^{\log s})$, where s is the number of leaves in the decision tree [1].

An even more interesting direction of research would be to obtain hardness results for learning by richer representation classes, such as AC^0 circuits, classes of neural networks and, ultimately, unrestricted circuits.

Cross-References

- ▶ Cryptographic Hardness of Learning
- ▶ Graph Coloring
- ▶ Learning DNF Formulas
- ▶ PAC Learning

Recommended Reading

1. Alekhnovich M, Braverman M, Feldman V, Klivans A, Pitassi T (2008) The complexity of properly learning simple classes. J Comput Syst Sci 74(1):16–34
2. Ben-David S, Eiron N, Long PM (2003) On the difficulty of approximately maximizing agreements. J Comput Syst Sci 66(3):496–514
3. Blum AL, Rivest RL (1992) Training a 3-node neural network is NP-complete. Neural Netw 5(1):117–127
4. Blumer A, Ehrenfeucht A, Haussler D, Warmuth M (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965
5. Bshouty N (1995) Exact learning via the monotone theory. Inform Comput 123(1):146–153
6. Feldman V, Gopalan P, Khot S, Ponuswami A (2009) On agnostic learning of parities, monomials and halfspaces. SIAM J Comput 39(2):606–645
7. Feldman V (2009) Hardness of approximate two-level logic minimization and pac learning with membership queries. J Comput Syst Sci 75(1):13–26
8. Feldman V, Guruswami V, Raghavendra P, Wu Y (2012) Agnostic learning of monomials by halfspaces is hard. SIAM J Comput 41(6):1558–1590
9. Garey M, Johnson DS (1979) Computers and intractability. W.H. Freeman, San Francisco
10. Guruswami V, Raghavendra P (2009) Hardness of learning halfspaces with noise. SIAM J Comput 39(2):742–765
11. Hancock T, Jiang T, Li M, Tromp J (1995) Lower bounds on learning decision lists and trees. In: Mayr

EW, Puech C (eds) STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, pp 527–538. Springer, Berlin/Heidelberg
12. Haussler D (1992) Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inform Comput 100(1):78–150
13. Jackson J (1997) An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J Comput Syst Sci 55:414–440
14. Kearns M, Schapire R, Sellie L (1994) Toward efficient agnostic learning. Mach Learn 17(2–3):115–141
15. Kearns M, Valiant L (1994) Cryptographic limitations on learning boolean formulae and finite automata. J ACM 41(1):67–95
16. Kearns M, Vazirani U (1994) An introduction to computational learning theory. MIT, Cambridge
17. Khot S, Saket R (2011) On the hardness of learning intersections of two halfspaces. J Comput Syst Sci 77(1):129–141
18. Pitt L, Valiant L (1988) Computational limitations on learning from examples. J ACM 35(4):965–984
19. Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Harmonic Algorithm for Online Bin Packing

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

Keywords

Bin packing; Bounded space algorithms; Competitive ratio

Years and Authors of Summarized Original Work

1985; Lee, Lee

Problem Definition

One of the goals of the design of the harmonic algorithm (or class of algorithms) was to provide an online algorithm for the classic bin packing problem that performs well with respect to the asymptotic competitive ratio, which is the standard measure for online algorithms for bin packing type problems. The competitive ratio for a given input is the ratio between the costs of the algorithm and of an optimal off-line solution. The asymptotic competitive ratio is the worst-case competitive ratio of inputs for which the optimal cost is sufficiently large. In the *online* (standard) bin packing problem, items of rational sizes in $(0, 1]$ are presented one by one. The algorithm must pack each item into a bin before the following item is presented. The total size of items packed into a bin cannot exceed 1, and the goal is to use the minimum number of bins, where a bin is used if at least one item was packed into it. All items must be packed, and the supply of bins is unlimited.

When an algorithm acts on an input, it can decide to *close* some of its bins and never use them again. A bin is called *closed* in such a case, while otherwise a used bin (which already has at least one item) is called *open*. The motivation for closing bins is to obtain fast running times per item (so that the algorithm will pack it into a bin selected out of a small number of options). Simple algorithms such as First Fit (FF), Best Fit (BF), and Worst Fit (WF) have worst-case running times of $O(\log N)$ per item, where N is the number of items at the time of assignment of the new item. On the other hand, the simple algorithm Next Fit (NF), which keeps at most of open bin and closes it when a new item cannot be packed there (before it uses a new bin for the new item), has a worst-case running time of $O(1)$ per item. Algorithms that keep a constant number of open bins are called *bounded space*. In many practical applications, this property is desirable, since the number of candidate bins for a new item is small and it does not increase with the input size.

Algorithm HARM$_k$ (for an integer $k \geq 3$) was defined by Lee and Lee [7]. The fundamental and natural idea of "harmonic-based" algorithms is classify each item by size first (for online algorithms, the classification of an item must be done immediately upon arrival) and then pack it according to its class (instead of letting the exact size influence packing decisions). For the classification of items, HARM$_k$ splits the interval $(0,1]$ into subintervals. There are $k-1$ subintervals of the form $(\frac{1}{i+1}, \frac{1}{i}]$ for $i = 1, \ldots, k-1$ and one final subinterval $(0, \frac{1}{k}]$. Each bin will contain only items from one subinterval (type). Every type is packed independently into its own bins using NF. Thus, there are at most $k-1$ open bins at each time (since for items of sizes above $\frac{1}{2}$, two items cannot share a bin, and any bin can be closed once it receives an item). Moreover, for $i < k$, as the items of type i have sizes no larger than $\frac{1}{i}$ but larger than $\frac{1}{i+1}$, every closed bin of this type will have exactly i items. For type k, a closed bin will contain at least k items, but it may contain many more items. This defines a class of algorithms (containing one algorithm for any $k \geq 3$). The term *the harmonic algorithm* (or simply HARM) refers to HARM$_k$ for a sufficiently large value of k, and its asymptotic competitive ratio is the infimum value that can be achieved as the asymptotic competitive ratio of any algorithm of this class.

Key Results

It was shown in paper [7] that for k tending to infinity, the asymptotic ratio of HARM is a sum of series denoted by Π_∞ (see below), and it is equal to approximately 1.69103. Moreover, this is the best possible asymptotic competitive ratio of any online bounded space algorithm for standard bin packing.

The crucial item sizes are of the form $\frac{1}{\ell} + \varepsilon$, where $\varepsilon > 0$ is small and ℓ is an integer. These are items of type $\ell - 1$, and bins consisting of such items contain $\ell - 1$ items (except for the last bin used for this type that may contain a smaller number of items). However, a bin (of an off-line solution) that already contains an item of size $\frac{1}{2} + \varepsilon_1$ and an item of size $\frac{1}{3} + \varepsilon_2$ (for some small $\varepsilon_1, \varepsilon_2 > 0$) cannot contain also an item whose size is slightly above $\frac{1}{4}$. The largest item of this form would be slightly larger than $\frac{1}{7}$. Thus, the following sequence was defined [7]. Let $\pi_1 = 1$ and, for $j > 1$, $\pi_j = \pi_{j-1}(\pi_{j-1}+1)$ (note that $\pi_{j'}$ is divisible by any π_j for $j < j'$). It turns out that the crucial item sizes are just above $\frac{1}{\pi_j + 1}$. The series $\sum_{j=1}^{\infty} \frac{1}{\pi_j}$ give the asymptotic competitive ratio of the HARM, Π_∞. For a long time the best lower bound on the asymptotic competitive ratio of (unbounded space) online algorithms was the one by van Vliet [8, 13], proved using this sequence (but the current best lower bound was proved using another set of inputs [1]).

In order to prove the upper bound Π_∞ on the competitive ratio, weights were used [12]. In this case weights are defined (for a specific value of k) quite easily such that all bins (except for the bins that remain open when the algorithm terminates) have total weights of at least 1. The weight of an item of type $i < k$ is $\frac{1}{i}$. The bins of type k are almost full for sufficiently large values of k (a bin can be closed only if the total size of its items exceeds $1 - \frac{1}{k}$). Assigning such an item a weight that is $\frac{k}{k-1}$ times its size will allow one to show that all bins except for a constant number of bins (at most $k-1$ bins) have total weights of at least 1. It is possible to show that the total weight of any packed bin is sufficiently close to Π_∞ for large values of k. As both HARM$_k$ and an optimal solution pack the same items, the competitive ratio is implied. To show the upper bound on the total weight of any packed bin, it is required to show that the worst-case bin contains exactly one item of size just above $\frac{1}{\pi_j + 1}$ for $\pi_j \leq k - 1$ (and the remaining space can only contain items of type k). Roughly speaking, this holds as once it was proved that the bin contains the largest such items, the largest possible additional weight can be obtained only by adding the next such item.

Proving that no better bounded space algorithms exist can be done as follows. Let j' be a fixed integer. Let N be a large integer and consider a sequence containing N items of each size $\frac{1}{\pi_j} + \delta$ for a sufficiently small $\delta > 0$, for any $j = j', j'-1, \cdots, 1$. If δ is chosen appropriately, we have $\sum_{j=1}^{j'} \frac{1}{\pi_j + 1} + j'\delta < 1$, so the items can be packed (off-line) into N bins. However, if items are presented in this order (sorted by nondecreasing size), after all items of one size have been presented, only a constant number of bins can receive larger items, and thus the items of each size are packed almost independently.

Related Results

The space of a bounded space algorithm is the number of open bins that it can have. The space of NF is 1, while the space of harmonic algorithms increases with k. A bounded space algorithm with space 2 and the same asymptotic competitive ratio as FF and BF have been designed [3] (for comparison, HARM_3 has an asymptotic competitive ratio of $\frac{7}{4}$). A modification where smaller space is used to obtain the same competitive ratios of harmonic algorithms (or alternatively, smaller competitive ratios were obtained using the same space) was designed by Woeginger [15]. Thus, there exists another sequence of bounded space algorithms, with an increasing sequence of open bins, where their sequence of competitive ratios tends to Π_∞ such that the space required for every competitive ratio is much smaller than that of [7].

One drawback of the model above is that an off-line algorithm can rearrange the items and does not have to process them as a sequence. The variant where it must process them in the same order as an online algorithm was studied as well [2]. Algorithms that are based on partitioning into classes and have smaller asymptotic competitive ratios (but they are obviously not bounded space) were designed [7, 9, 11].

Generalizations have been studied too, in particular, bounded space bin packing with cardinality constraints (where an item cannot receive more than t items for a fixed integer $t \geq 2$) [5], parametric bin packing (where there is an upper bound strictly smaller than 1 on item sizes) [14], bin packing with rejection (where an item i has a rejection penalty r_i associated with it, and it can be either packed, or rejected for the cost r_i) [6], variable-sized bin packing (where bins of multiple sizes are available for packing) [10], and bin packing with resource augmentation (where the online algorithm can use bins of size $b > 1$ for a fixed rational number b, while an off-line algorithm still uses bins of size 1) [4]. In this last variant, the sequences of critical item sizes were redefined as a function of b, while variable-sized bin packing required a more careful partition into intervals.

Cross-References

▶ Current Champion for Online Bin Packing

Recommended Reading

1. Balogh J, Békési J, Galambos G (2012) New lower bounds for certain classes of bin packing algorithms. Theor Comput Sci 440–441:1–13
2. Chrobak M, Sgall J, Woeginger GJ (2011) Two-bounded-space bin packing revisited. In: Proceedings of the 19th annual European symposium on algorithms (ESA2011), Saarbrücken, Germany, pp 263–274
3. Csirik J, Johnson DS (2001) Bounded space on-line bin packing: best is better than first. Algorithmica 31:115–138
4. Csirik J, Woeginger GJ (2002) Resource augmentation for online bounded space bin packing. J Algorithms 44(2):308–320
5. Epstein L (2006) Online bin packing with cardinality constraints. SIAM J Discret Math 20(4):1015–1030
6. Epstein L (2010) Bin packing with rejection revisited. Algorithmica 56(4):505–528
7. Lee CC, Lee DT (1985) A simple online bin packing algorithm. J ACM 32(3):562–572
8. Liang FM (1980) A lower bound for on-line bin packing. Inf Process Lett 10(2):76–79
9. Ramanan P, Brown DJ, Lee CC, Lee DT (1989) On-line bin packing in linear time. J Algorithms 10:305–326
10. Seiden SS (2001) An optimal online algorithm for bounded space variable-sized bin packing. SIAM J Discret Math 14(4):458–470
11. Seiden SS (2002) On the online bin packing problem. J ACM 49(5):640–671
12. Ullman JD (1971) The performance of a memory allocation algorithm. Technical report 100, Princeton University, Princeton
13. van Vliet A (1992) An improved lower bound for online bin packing algorithms. Inf Process Lett 43(5):277–284
14. van Vliet A (1996) On the asymptotic worst case behavior of Harmonic Fit. J Algorithms 20(1):113–136
15. Woeginger GJ (1993) Improved space for bounded-space online bin packing. SIAM J Discret Math 6(4):575–581

Hierarchical Self-Assembly

David Doty
Computing and Mathematical Sciences,
California Institute of Technology, Pasadena,
CA, USA

Keywords

Hierarchical assembly; Intrinsic universality; Running time; Self-assembly; Verification

Years and Authors of Summarized Original Work

2005; Aggarwal, Cheng, Goldwasser, Kao, Espanes, Schweller
2012; Chen, Doty
2013; Cannon, Demaine, Demaine, Eisenstat, Patitz, Schweller, Summers, Winslow

Problem Definition

The general idea of *hierarchical* self-assembly (a.k.a., *multiple tile* [2], *polyomino* [8, 10], *two-handed* [3, 5, 6]) is to model self-assembly of tiles in which attachment of two multi-tile assemblies is allowed, as opposed to all attachments being that of a single tile onto a larger assembly. Several problems concern comparing hierarchical self-assembly to its single-tile-attachment variant (called the "seeded" model of self-assembly), so we define both models here. The model of hierarchical self-assembly was first defined (in a slightly different form that restricted the size of assemblies that could attach) by Aggarwal, Cheng, Goldwasser, Kao, Moisset de Espanes, and Schweller [2]. Several generalizations of the model exist that incorporated staged mixing of test tubes, "dissolvable" tiles, active signaling across tiles, etc., but here we restrict attention to the model closest to the seeded model of Winfree [9], different from that model only in the absence of a seed and the ability of two large assemblies to attach.

Supported by NSF grants CCF-1219274, CCF-1162589, and 1317694.

Definitions

A *tile type* is a unit square with four sides, each consisting of a *glue label* (often represented as a finite string) and a nonnegative integer *strength*. We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. An *assembly* is a positioning of tiles on the integer lattice $\mathbb{Z}^2$, i.e., a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. We write $|\alpha|$ to denote $|\text{dom } \alpha|$. Write $\alpha \sqsubseteq \beta$ to denote that α is a *subassembly* of β, which means that $\text{dom } \alpha \subseteq \text{dom } \beta$ and $\alpha(p) = \beta(p)$ for all points $p \in \text{dom } \alpha$. We abuse notation and take a tile type t to be equivalent to the single-tile assembly containing only t (at the origin if not otherwise specified). Two adjacent tiles in an assembly *interact* if the glue labels on their abutting sides are equal and have positive strength. Each assembly induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The assembly is τ*-stable* if every cut of its binding graph has strength at least τ, where the weight of an edge is the strength of the glue it represents. That is, the assembly is stable if at least energy τ is required to separate the assembly into two parts.

We now define both the seeded and hierarchical variants of the tile assembly model. A *seeded tile system* is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is a finite, τ-stable *seed assembly*, and τ is the *temperature*. If $\mathcal{T}$ has a single seed tile $s \in T$ (i.e., $\sigma(0,0) = s$ for some $s \in T$ and is undefined elsewhere), then we write $\mathcal{T} = (T, s, \tau)$. Let $|\mathcal{T}|$ denote $|T|$. An assembly α is *producible* if either $\alpha = \sigma$ or if β is a producible assembly and α can be obtained from β by the stable binding of a single tile. In this case, write $\beta \to_1 \alpha$ (α is producible from β by the attachment of one tile), and write $\beta \to \alpha$ if $\beta \to_1^* \alpha$ (α is producible from β by the attachment of zero or more tiles). An assembly is *terminal* if no tile can be τ-stably attached to it.

A *hierarchical tile system* is a pair $\mathcal{T} = (T, \tau)$, where T is a finite set of tile types and $\tau \in \mathbb{N}$

is the temperature. An assembly is *producible* if either it is a single tile from T or it is the τ-stable result of translating two producible assemblies without overlap. Therefore, if an assembly α is producible, then it is produced via an *assembly tree*, a full binary tree whose root is labeled with α, whose $|\alpha|$ leaves are labeled with tile types, and each internal node is a producible assembly formed by the stable attachment of its two child assemblies. An assembly α is *terminal* if for every producible assembly β, α and β cannot be τ-stably attached. If α can grow into β by the attachment of zero or more assemblies, then we write $\alpha \to \beta$.

In either model, let $\mathcal{A}[\mathcal{T}]$ be the set of producible assemblies of $\mathcal{T}$, and let $\mathcal{A}_\Box[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ be the set of producible, terminal assemblies of $\mathcal{T}$. A TAS $\mathcal{T}$ is *directed* (a.k.a., *deterministic*, *confluent*) if $|\mathcal{A}_\Box[\mathcal{T}]| = 1$. If $\mathcal{T}$ is directed with unique producible terminal assembly α, we say that $\mathcal{T}$ *uniquely produces* α. It is easy to check that in the seeded aTAM, $\mathcal{T}$ uniquely produces α if and only if every producible assembly $\beta \sqsubseteq \alpha$. In the hierarchical model, a similar condition holds, although it is more complex since hierarchical assemblies, unlike seeded assemblies, do not have a "canonical translation" defined by the seed position. $\mathcal{T}$ uniquely produces α if and only if for every producible assembly β, there is a translation β' of β such that $\beta' \sqsubseteq \alpha$. In particular, if there is a producible assembly $\beta \neq \alpha$ such that dom α = dom β, then α is not uniquely produced. Since dom β = dom α, every nonzero translation of β has some tiled position outside of dom α, whence no such translation can be a subassembly of α, implying α is not uniquely produced.

Power of Hierarchical Assembly Compared to Seeded

One sense in which we can conclude that one model of computation M is at least as powerful as another model of computation M' is to show that any machine defined by M' can be "simulated efficiently" by a machine defined by M. In self-assembly, there is a natural definition of what it means for one tile system $\mathcal{S}$ to "simulate" another $\mathcal{T}$. We now discuss intuitively how to define such a notion. There are several intricacies to the full formal definition that are discussed in further detail in [3, 5].

First, we require that there is a constant $k \in \mathbb{Z}^+$ (the "resolution loss") such that each tile type t in $\mathcal{T}$ is "represented" by one or more $k \times k$ blocks β of tiles in $\mathcal{S}$. In this case, we write $r(\beta) = t$, where $\beta : \{1, \ldots, k\}^2 \dashrightarrow S$ and S is the tile set of $\mathcal{S}$. Then β represents a $k \times k$ block of such tiles, possibly with empty positions at points $\mathbf{x}$ where $\beta(\mathbf{x})$ is undefined. We call such a $k \times k$ block in $\mathcal{S}$ a "macrotile." We can extend r to a function R that, given an assembly $\alpha_{\mathcal{S}}$ partitioned into $k \times k$ macrotiles, outputs an assembly $\alpha_{\mathcal{T}}$ of $\mathcal{T}$ such that, for each macrotile β of $\alpha_{\mathcal{S}}$, $r(\beta) = t$, where t is the tile type at the corresponding position in $\alpha_{\mathcal{T}}$.

Given such a representation function R indicating how to interpret assemblies of $\mathcal{S}$ as representing assemblies of $\mathcal{T}$, we now define what it means to say that $\mathcal{S}$ *simulates* $\mathcal{T}$. For each producible assembly $\alpha_{\mathcal{T}}$ of $\mathcal{T}$, there is a producible assembly $\alpha_{\mathcal{S}}$ of $\mathcal{S}$ such that $R(\alpha_{\mathcal{S}}) = \mathcal{T}$, and furthermore, for every producible assembly $\alpha_{\mathcal{S}}$, if $R(\alpha_{\mathcal{S}}) = \mathcal{T}$, then $\mathcal{T}$ is producible in $\mathcal{T}$. Finally, we require that R respects the "single attachment" dynamics of $\mathcal{T}$: there is a single tile that can be attached to $\alpha_{\mathcal{T}}$ to result in $\alpha'_{\mathcal{T}}$ if and only if there is some sequence of attachments to $\alpha_{\mathcal{S}}$ that results in assembly $\alpha'_{\mathcal{S}}$ such that $R(\alpha'_{\mathcal{S}}) = \alpha'_{\mathcal{T}}$.

With such an idea in mind, we can ask, "Is the hierarchical model at least as powerful as the seeded model?"

Problem 1 For every seeded tile system $\mathcal{T}$, design a hierarchical tile system $\mathcal{S}$ that simulates $\mathcal{T}$.

Another interpretation of a solution to Problem 1 is that, to the extent that the hierarchical model is more realistic than the seeded model by incorporating the reality that tiles may aggregate even in the absence of a seed, such a solution shows how to enforce seeded growth even in such an unfriendly environment that permits non-seeded growth.

Assembly Time

We now define time complexity for hierarchical systems (this definition first appeared in [4], where it is explained in more detail). We treat each assembly as a single molecule. If two assemblies α and β can attach to create an assembly γ, then we model this as a chemical reaction $\alpha + \beta \to \gamma$, in which the rate constant is assumed to be equal for all reactions (and normalized to 1). In particular, if α and β can be attached in two different ways, this is modeled as two different reactions, even if both result in the same assembly.

At an intuitive level, the model we define can be explained as follows. We imagine dumping all tiles into solution at once, and at the same time, we grab one particular tile and dip it into the solution as well, pulling it out of the solution when it has assembled into a terminal assembly. Under the seeded model, the tile we grab will be a seed, assumed to be the only copy in solution (thus requiring that it appears only once in any terminal assembly). In the seeded model, no reactions occur other than the attachment of individual tiles to the assembly we are holding. In the hierarchical model, other reactions are allowed to occur in the background (we model this using the standard mass-action model of chemical kinetics [7]), but only those reactions with the assembly we are holding move it "closer" to completion. The other background reactions merely change concentrations of other assemblies (although these indirectly affect the time it will take our chosen assembly to complete, by changing the rate of reactions with our chosen assembly).

More formally, let $\mathcal{T} = (T, \tau)$ be a hierarchical TAS, and let $\rho : T \to [0,1]$ be a concentrations function, giving the *initial* concentration of each tile type (we require that $\sum_{t \in T} \rho(t) = 1$, a condition known as the "finite density constraint"). Let $\mathbb{R}^+ = [0, \infty)$, and let $t \in \mathbb{R}^+$. For $\alpha \in \mathcal{A}[\mathcal{T}]$, let $[\alpha]_\rho(t)$ (abbreviated $[\alpha](t)$ when ρ is clear from context) denote the concentration of α at time t with respect to initial concentrations ρ, defined as follows. Given two assemblies α and β that can attach to form γ, we model this event as a chemical reaction $R : \alpha + \beta \to \gamma$. Say that a reaction $\alpha + \beta \to \gamma$ is *symmetric* if $\alpha = \beta$. Define the *propensity* (a.k.a., *reaction rate*) of R at time $t \in \mathbb{R}^+$ to be $\rho_R(t) = [\alpha](t) \cdot [\beta](t)$ if R is not symmetric and $\rho_R(t) = \frac{1}{2} \cdot [\alpha](t)^2$ if R is symmetric.

If α is consumed in reactions $\alpha + \beta_1 \to \gamma_1, \ldots, \alpha + \beta_n \to \gamma_n$ and produced in asymmetric reactions $\beta'_1 + \gamma'_1 \to \alpha, \ldots, \beta'_m + \gamma'_m \to \alpha$ and symmetric reactions $\beta''_1 + \beta''_1 \to \alpha, \ldots, \beta''_p + \beta''_p \to \alpha$, then the concentration $[\alpha](t)$ of α at time t is described by the differential equation:

$$\frac{d[\alpha](t)}{dt} = \sum_{i=1}^{m} [\beta'_i](t) \cdot [\gamma'_i](t) + \sum_{i=1}^{p} \frac{1}{2} \cdot [\beta''_i](t)^2 - \sum_{i=1}^{n} [\alpha](t) \cdot [\beta_i](t), \qquad (1)$$

with boundary conditions $[\alpha](0) = \rho(r)$ if α is an assembly consisting of a single tile r and $[\alpha](0) = 0$ otherwise. In other words, the propensities of the various reactions involving α determine its rate of change, negatively if α is consumed and positively if α is produced.

This completes the definition of the dynamic evolution of concentrations of producible assemblies; it remains to define the time complexity of assembling a terminal assembly. Although we have distinguished between seeded and hierarchical systems, for the purpose of defining a model of time complexity in hierarchical systems and comparing them to the seeded system time complexity model of [1], it is convenient to introduce a seedlike "timekeeper tile" into the hierarchical system, in order to stochastically analyze the growth of this tile when it reacts in a solution that is itself evolving according to the continuous model described above. The seed does not have the purpose of nucleating growth but is introduced merely to focus attention on a single molecule that has not yet assembled anything, in order to ask how long it will take to assemble into a terminal assembly. The choice of which tile type to pick will be a parameter of the definition, so that a system may have different assembly times depending on the choice of timekeeper tile.

Fix a copy of a tile type s to designate as a "timekeeper seed." The assembly of s into some terminal assembly $\hat{\alpha}$ is described as a time-dependent continuous-time Markov process in which each state represents a producible assembly containing s, and the initial state is the size-1 assembly with only s. For each state α representing a producible assembly with s at the origin, and for each pair of producible assemblies β, γ such that $\alpha + \beta \to \gamma$ (with the translation assumed to happen only to β so that α stays "fixed" in position), there is a transition in the Markov process from state α to state γ with transition rate $[\beta](t)$.

We define $\mathbf{T}_{\mathcal{T},\rho,s}$ to be the random variable representing the time taken for the copy of s to assemble into a terminal assembly via some sequence of reactions as defined above. We define the time complexity of a directed hierarchical TAS $\mathcal{T}$ with concentrations ρ and timekeeper s to be $\mathsf{T}(\mathcal{T},\rho,s) = \mathrm{E}\left[\mathbf{T}_{\mathcal{T},\rho,s}\right]$.

For a shape $S \subset \mathbb{Z}^2$ (finite and connected), define the *diameter* of S to be $\mathrm{diam}(S) = \max\limits_{\mathbf{u},\mathbf{v}\in S} \|\mathbf{u}-\mathbf{v}\|_1$, where $\|\mathbf{w}\|_1$ is the L_1 norm of $\mathbf{w}$.

Problem 2 Design a hierarchical tile system $\mathcal{T} = (T,\tau)$ such that every producible terminal assembly $\hat{\alpha}$ has the same shape S, and for some $s \in T$ and concentrations function $\rho : T \to [0,1]$, $\mathsf{T}(\mathcal{T},\rho,s) = o(\mathrm{diam}(S))$.

It is provably impossible to achieve this with the seeded model [1, 4], since all assemblies in that model require expected time at least proportional to their diameter.

Key Results

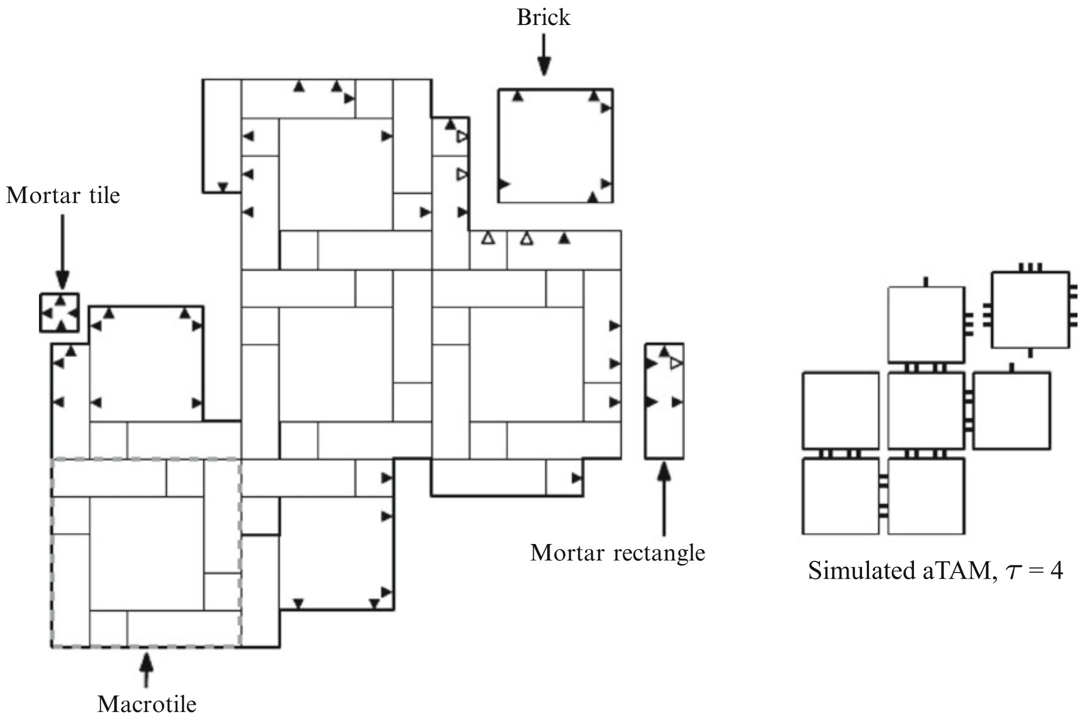
Power of Hierarchical Assembly Compared to Seeded

Cannon, Demaine, Demaine, Eisenstat, Patitz, Schweller, Summers, and Winslow [3] showed a solution to Problem 1. (They also showed several other ways in which the hierarchical model is more powerful than the seeded model, but we restrict attention to simulation here.) For the most part, temperature 2 seeded systems are as powerful as those at higher temperatures, but the simulation results of [3] hold for higher temperatures as well. In particular, they showed that every seeded temperature ≥ 4 tile system $\mathcal{T}$ can be simulated by a hierarchical temperature 4 tile system (as well as showing it is possible for temperature τ hierarchical tile systems to simulate temperature τ seeded tile systems for $\tau \in \{2,3\}$, using similar logic to the higher-temperature construction). The definition of simulation has a parameter k indicating the resolution loss of the simulation. In fact, the simulation described in [3] requires only resolution loss $k = 5$.

Figure 1 shows an example of $\mathcal{S}$ simulating $\mathcal{T}$. The construction enforces the "simulation of dynamics" constraint that if and only if a single tile can attach in $\mathcal{T}$, and then a 5×5 macrotile representing it in $\mathcal{S}$ can assemble. It is critical that each tile type in $\mathcal{T}$ is represented by *more than one* type of macrotile in $\mathcal{S}$: each different type of macrotile represents a different subset of sides that can cooperate to allow the tile to bind. To achieve this, each macrotile consists of a central "brick" (itself a 3×3 block composed of 9 unique tile types with held together with strength-4 glues) surrounded by "mortar" (forming a ring around the central brick). Figure 1 shows "mortar rectangles" but, similarly to the brick, these are just 3×1 assemblies of 3 individual tile types with strength-4 glues. The logic of the system is such that if a brick B designed for a subset of cooperating sides $C \subseteq \{\mathsf{N},\mathsf{S},\mathsf{E},\mathsf{W}\}$, then only if the mortar for all sides in C is present can B attach. Its attachment is required to fill in the remaining mortar representing the other sides in $\{\mathsf{N},\mathsf{S},\mathsf{E},\mathsf{W}\} \setminus C$ that may not be present. Finally, those tiles enable the assembly of mortar in *adjacent* 5×5 blocks, to be ready for possible cooperation to bind bricks in those blocks.

Assembly Time

Chen and Doty [4] showed a solution to Problem 2, by proving that for infinitely many $n \in \mathbb{N}$, there is a (non-directed) hierarchical TAS



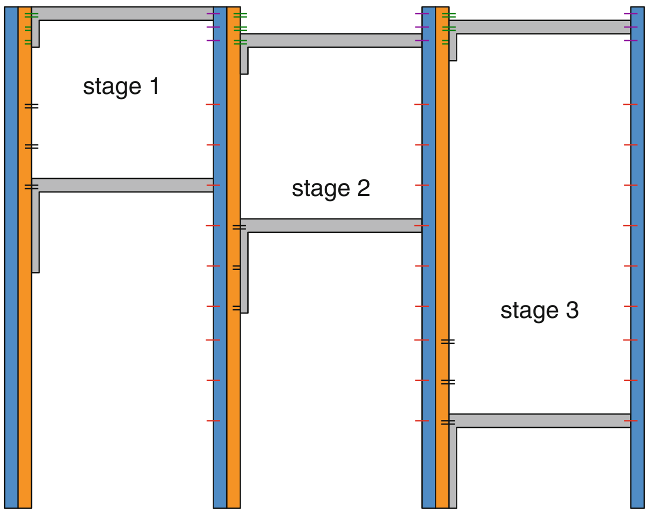
Hierarchical Self-Assembly, Fig. 1 Simulation of a seeded tile system $\mathcal{T}$ of temperature ≥ 4 by a hierarchical tile system $\mathcal{S}$ of temperature 4 (Figure taken from [3]). *Filled arrows* represent glues of strength 2, and *unfilled arrows* represent glues of strength 1. In the seeded tile system, the number of *dashes* on the side of a tile represent its strength

$\mathcal{T} = (T, 2)$ that strictly self-assembles an $n \times n'$ rectangle S, where $n' = o(n)$ (hence $\mathrm{diam}(S) = \Theta(n)$), such that $|T| = O(\log n)$ and there is a tile type $s \in T$ and concentrations function $\rho : T \to [0, 1]$ such that $\mathsf{T}(\mathcal{T}, \rho, s) = O(n^{4/5} \log n)$.

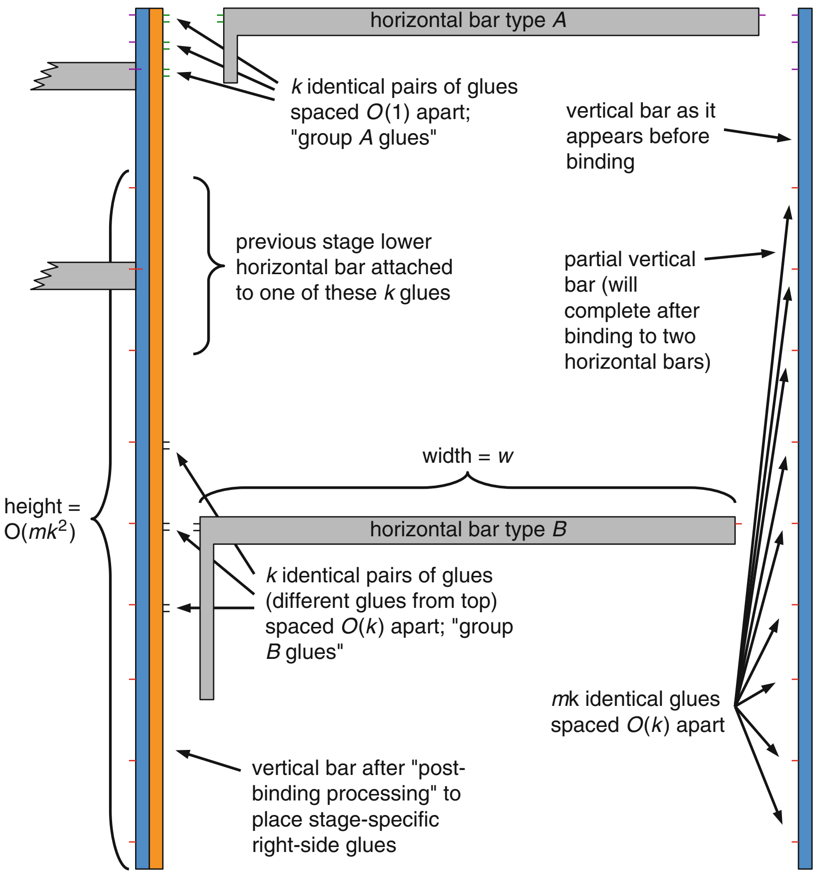
The construction consists of $m = n^{1/5}$ stages shown in Fig. 2, where each stage consists of the attachment of two "horizontal bars" to a single "vertical bar" as shown in Fig. 3. The vertical bar of the next stage then attaches to the right of the two horizontal bars, which cooperate to allow the binding because they each have a single strength 1 glue. All vertical bars are identical when they attach, but attachment triggers the growth of some tiles (shown in orange in Figs. 2 and 3) that make the attachment sites on the right side different from their locations in the previous stage, which is how the stages "count down" from m to 1.

The bars themselves are assembled in a "standard" way that requires time linear in the diameter of the bar, which is $w = n^{4/5}$ for a horizontal bar and $mk^2 = n^{3/5}$ (where k is a parameter that we set to be $n^{1/5}$) for a vertical bar. The speedup comes from the fact that each horizontal bar can attach to one of k different binding sites on a vertical bar, so the expected time for this to happen is factor k lower than if there were only a single binding site. The vertical "arm" on the left of each horizontal bar has the purpose of preventing any other horizontal bars from binding near it. Each stage also requires filler tiles to fill in the gap regions, but the time required for this is negligible compared to the time for all vertical and horizontal bars to attach.

Note that this construction is not directed: although every producible terminal assembly has the shape of an $n \times n'$ rectangle, there are many such terminal assemblies. Chen and Doty [4] also showed that for a class of directed systems called "partial order tile systems," no solution to Problem 2 exists: provably any such tile system



Hierarchical Self-Assembly, Fig. 2 High-level overview of interaction of "vertical bars" and "horizontal bars" to create the rectangle in the solution to Problem 2 that assembles in time sublinear in its diameter. Filler tiles fill in the empty regions. If glues overlap two regions then represent a formed bond. If glues overlap one region but not another, they are glues from the former region but are mismatched (and thus "covered and protected") by the latter region


horizontal bar type *A*
k identical pairs of glues spaced *O*(1) apart; "group *A* glues"
vertical bar as it appears before binding
previous stage lower horizontal bar attached to one of these *k* glues
partial vertical bar (will complete after binding to two horizontal bars)
width = *w*
height = O(*mk*²)
horizontal bar type *B*
k identical pairs of glues (different glues from top) spaced *O*(*k*) apart; "group *B* glues"
*m*k identical glues spaced *O*(*k*) apart
vertical bar after "post-binding processing" to place stage-specific right-side glues

Hierarchical Self-Assembly, Fig. 3 "Vertical bars" for the construction of a fast-assembling square, and their interaction with horizontal bars, as shown for a single stage of Fig. 2. "Type *B*" horizontal bars have a longer vertical arm than "Type *A*" since the glues they must block are farther apart assembling a shape of diameter d requires expected time $\Omega(d)$.

Open Problems

It is known [2] that the tile complexity of assembling an $n \times k$ rectangle in the seeded aTAM, if $k < \frac{\log n}{\log\log n - \log\log\log n}$, is asymptotically lower bounded by $\Omega\left(\frac{n^{1/k}}{k}\right)$ and upper bounded by $O(n^{1/k})$. For the hierarchical model, the upper bound holds as well [2], but the strongest known lower bound is the information-theoretic $\Omega\left(\frac{\log n}{\log\log n}\right)$.

Question 1 What is the tile complexity of assembling an $n \times k$ rectangle in the hierarchical model, when $k < \frac{\log n}{\log\log n - \log\log\log n}$?

Cross-References

- ▸ Experimental Implementation of Tile Assembly
- ▸ Patterned Self-Assembly Tile Set Synthesis
- ▸ Robustness in Self-Assembly
- ▸ Self-Assembly at Temperature 1
- ▸ Self-Assembly of Fractals
- ▸ Self-Assembly with General Shaped Tiles
- ▸ Staged Assembly
- ▸ Temperature Programming in Self-Assembly

Recommended Reading

1. Adleman LM, Cheng Q, Goel A, Huang M-D (2001) Running time and program size for self-assembled squares. In: STOC 2001: proceedings of the thirty-third annual ACM symposium on theory of computing, Hersonissos. ACM, pp 740–748
2. Aggarwal G, Cheng Q, Goldwasser MH, Kao M-Y, Moisset de Espanés P, Schweller RT (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34:1493–1515. Preliminary version appeared in SODA 2004
3. Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller RT, Summers SM, Winslow A (2013) Two hands are better than one (up to constant factors). In: STACS 2013: proceedings of the thirtieth international symposium on theoretical aspects of computer science, Kiel, pp 172–184
4. Chen H-L, Doty D (2012) Parallelism and time in hierarchical self-assembly. In: SODA 2012: proceedings of the 23rd annual ACM-SIAM symposium on discrete algorithms, Kyoto, pp 1163–1182
5. Demaine ED, Patitz MJ, Rogers T, Schweller RT, Summers SM, Woods D (2013) The two-handed tile assembly model is not intrinsically universal. In: ICALP 2013: proceedings of the 40th international colloquium on automata, languages and programming, Riga, July 2013
6. Doty D, Patitz MJ, Reishus D, Schweller RT, Summers SM (2010) Strong fault-tolerance for self-assembly with fuzzy temperature. In: FOCS 2010: proceedings of the 51st annual IEEE symposium on foundations of computer science, Las Vegas, pp 417–426
7. Epstein IR, Pojman JA (1998) An introduction to nonlinear chemical dynamics: oscillations, waves, patterns, and chaos. Oxford University Press, Oxford
8. Luhrs C (2010) Polyomino-safe DNA self-assembly via block replacement. Nat Comput 9(1):97–109. Preliminary version appeared in DNA 2008
9. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology, June 1998
10. Winfree E (2006) Self-healing tile sets. In: Chen J, Jonoska N, Rozenberg G (eds) Nanotechnology: science and computation. Natural computing series. Springer, Berlin/New York, pp 55–78

Hierarchical Space Decompositions for Low-Density Scenes

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands

Keywords

Binary space partitions; Compressed quadtrees; Computational geometry; Hierarchical space decompositions; Realistic input models

Years and Authors of Summarized Original Work

2000; De Berg
2010; De Berg, Haverkort, Thite, Toma

Problem Definition

Many algorithmic problems on spatial data can be solved efficiently if a suitable decomposition of the ambient space is available. Two desirable properties of the decomposition are that its cells have a nice shape – convex and/or of constant complexity – and that each cell intersects only a few objects from the given data set. Another desirable property is that the decomposition is hierarchical, meaning that the space is partitioned in a recursive manner. Popular hierarchical space decompositions include quadtrees and binary space partitions.

When the objects in the given data set are nonpoint objects, they can be fragmented by the partitioning process. This fragmentation has a negative impact on the storage requirements of the decomposition and on the efficiency of algorithms operating on it. Hence, it is desirable to minimize fragmentation. In this chapter, we describe methods to construct linear-size compressed quadtrees and binary space partitions for so-called low-density sets. To simplify the presentation, we describe the constructions in the plane. We use S to denote the set of n objects for which we want to construct a space decomposition and assume for simplicity that the objects in S are disjoint, convex, and of nonzero area.

Binary Space Partitions

A *binary space partition* for a set S of n objects in the plane is a recursive decomposition of the plane by lines, typically such that each cell in the final decomposition intersects only a few objects from S. The tree structure modeling this decomposition is called a *binary space partition tree*, or BSP *tree* for short – see Fig. 1 for an illustration. Thus, a BSP tree $\mathcal{T}$ for S can be defined as follows.

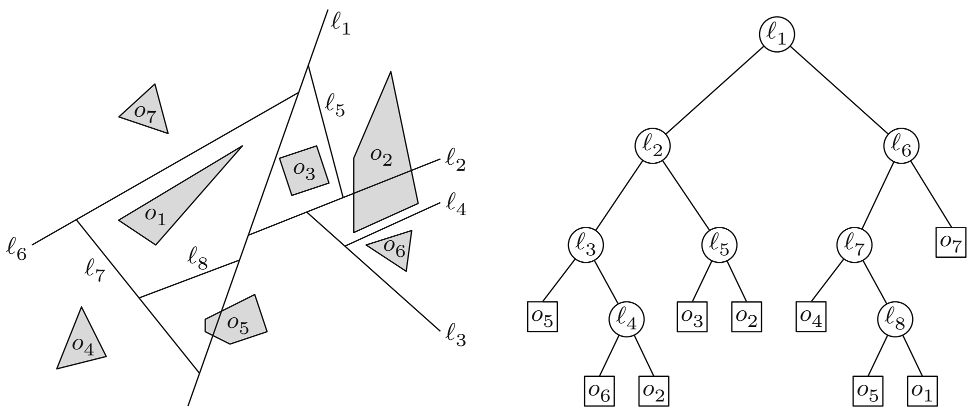
- If a predefined stopping criterion is met – often this is when $|S|$ is sufficiently small – then $\mathcal{T}$ consists of a single leaf where the set S is stored.
- Otherwise the root node v of $\mathcal{T}$ stores a suitably chosen *splitting line* ℓ. Let ℓ^- and ℓ^+ denote the half-planes lying to the left and to the right of ℓ, respectively (or, if ℓ is horizontal, below and above ℓ).
  - The left subtree of v is a BSP tree for $S^- := \{o \cap \ell^- : o \in S\}$, the set of object fragments lying in the half-plane ℓ^-.
  - The right subtree of v is a BSP tree for $S^+ := \{o \cap \ell^+ : o \in S\}$, the set of object fragments lying in the half-plane ℓ^+.

The *size* of a BSP tree is the total number of object fragments stored in the tree.

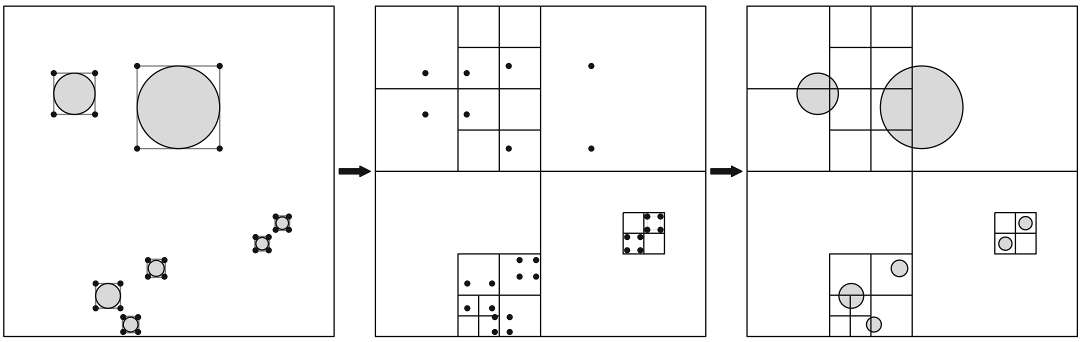
Compressed Quadtrees

Let $U = [0, 1]^2$ be the unit square. We say that a square $\sigma \subseteq U$ is a *canonical square* if there is an integer $k \geqslant 0$ such that σ is a cell of the regular subdivision of U into $2^k \times 2^k$ squares. A *donut* is the set-theoretic difference $\sigma_{\text{out}} \setminus \sigma_{\text{in}}$ of a canonical square σ_{out} and a canonical square $\sigma_{\text{in}} \subset \sigma_{\text{out}}$. A *compressed quadtree* $\mathcal{T}$ for a set P of points inside a canonical square σ defined as follows; see also Fig. 2 (middle).

- If a predefined stopping criterion is met – usually this is when $|P|$ is sufficiently small – then $\mathcal{T}$ consists of a single leaf storing the set P.
- If the stopping criterion is not met, then $\mathcal{T}$ is defined as follows. Let σ_{NE} denote the north-east quadrant of σ and let $P_{\text{NE}} := P \cap \sigma_{\text{NE}}$. Define σ_{SE}, σ_{SW}, σ_{NW} and P_{SE}, P_{SW}, P_{NW} similarly for the other three quadrants. (Here we should make sure that points on the boundary between quadrants are assigned to quadrants in a consistent manner.) Now $\mathcal{T}$ consists of a root node v with four or two children, depending on how many of the sets P_{NE}, P_{SE}, P_{SW}, P_{NW} are nonempty:
  - If at least two of the sets P_{NE}, P_{SE}, P_{SW}, P_{NW} are nonempty, then v has four children v_{NE}, v_{SE}, v_{SW}, v_{NW}. The child v_{NE} is the root of a compressed quadtree for the set P_{NE} inside the square σ_{NW}; the other three children are defined similarly for the point sets inside the other quadrants.
  - If only one of P_{NE}, P_{SE}, P_{SW}, P_{NW} is nonempty, then v has two children v_{in} and v_{out}. The child v_{in} is the root of a



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 1 A binary space partition for a set of polygons (*left*) and the corresponding BSP tree (*right*)



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 2 Construction of a compressed quadtree for a set of disks: take the bounding-box vertices (*left*), construct a compressed quadtree for the vertices (*middle*), and put the disks back in (*right*)

compressed quadtree for P inside σ_{in}, where σ_{in} is the smallest canonical square containing all points from P. The other child is a leaf corresponding to the donut $\sigma \setminus \sigma_{\text{in}}$.

A compressed quadtree for a set of n points has size $O(n)$.

Above we defined compressed quadtrees for point sets. In this chapter, we are interested in compressed quadtrees for nonpoint objects. These are defined similarly: each internal node corresponds to a canonical square, and each leaf is a canonical square or a donut. This time donuts need not be empty, but may intersect objects (although not too many). The right picture in Fig. 2 shows a compressed quadtree for a set of disks. The *size* of a compressed quadtree for a set of nonpoint objects is defined as the total number of object fragments stored in the tree. Because nonpoint objects may be split into fragments during the subdivision process, a compressed quadtree for nonpoint objects is not guaranteed to have linear size.

Low-Density Scenes

The main question we are interested in is the following: given a set S of n objects, can we construct a compressed quadtree or BSP tree with $O(n)$ leaves such that each leaf region intersects $O(1)$ objects? In general, the answer to this question is no. For compressed quadtrees, this can be seen by considering a set S of slanted parallel segments that are very close together. A linear-size BSP tree cannot be guaranteed either: there are sets of n disjoint segments in the plane for which any BSP tree has size $\Omega(n \log n / \log\log n)$ [7]. In $\mathbb{R}^3$ the situation is even worse: there are sets of n disjoint triangles for which any BSP tree has size $\Omega(n^2)$ [5]. (Both bounds are tight: there are algorithms that guarantee a BSP tree of size $O(n \log n / \log\log n)$ in the plane [8] and of size $O(n^2)$ in $\mathbb{R}^3$ [6].) Fortunately, in practice, the objects for which we want to construct a space decomposition are often distributed nicely, which allows us to construct much smaller decompositions than for the worst-case examples mentioned above. To formalize this, we define the concept of *density* of a set of objects in $\mathbb{R}^d$.

Definition 1 The *density* of a set S of objects in $\mathbb{R}^d$, denoted density(S), is defined as the smallest number λ such that the following holds: any ball $b \subset \mathbb{R}^d$ intersects at most λ objects $o \in S$ such that diam$(o) \geqslant$ diam(b), where diam$(\cdot)$ denotes the diameter of an object.

As illustrated in Fig. 3(i), a set of n parallel segments can have density n if the segments are very close together. In most practical situations, however, the input objects are distributed nicely and the density will be small. For many classes of objects, one can even prove that the density is $O(1)$. For example, a set of disjoint disks in the plane has density at most 5. More generally, any set of disjoint objects that are *fat* – examples of fat objects are disks, squares, triangles whose minimum angle is lower bounded – has density $O(1)$ [3]. The main question now is: Is low density sufficient to guarantee a hierarchical space decomposition of linear size? The answer is yes, and constructing the space decomposition is surprisingly easy.

Key Results

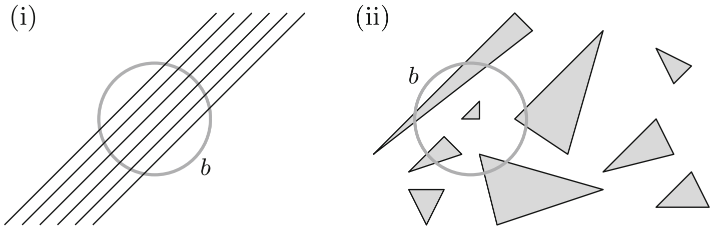
The construction of space decompositions for low-density sets is based on the following lemma. In the lemma, the square σ is considered to be open, that is, σ does not include its boundary. Let bb(o) denote the axis-aligned bounding box of an object o.

Lemma 1 *Let S be a set of n objects in the plane and let B_S denote the set of $4n$ vertices of the bounding boxes* bb(o) *of the objects $o \in S$. Let σ be any square region in the plane. Then the number of objects in S intersecting σ is at most $k + 4\lambda$, where k is the number of bounding-box vertices inside σ and $\lambda :=$* density(S).

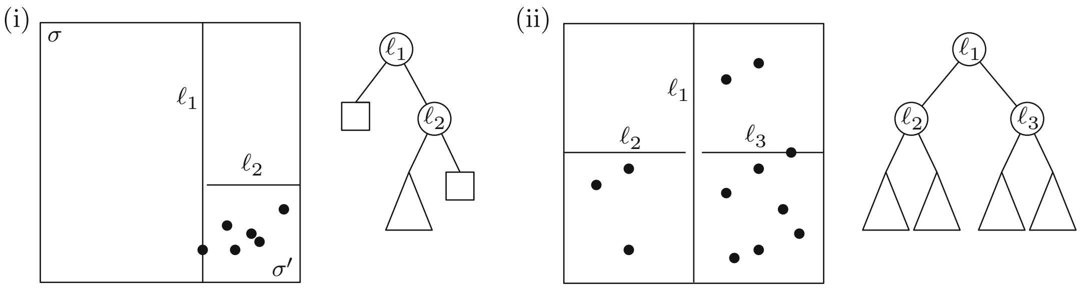
With Lemma 1 in hand, it is surprisingly simple to construct BSP trees or compressed quadtrees of small size for any given set S whose density is small.

Binary Space Partitions

For BSP trees we proceed as follows. Let B_S be the set of vertices of the bounding boxes of the objects in S. In a generic step in the recursive construction of $\mathcal{T}$, we are given a square σ and the set of points $B_S(\sigma) := B_S \cap \sigma$. Initially σ is a square containing all points of B_S. When $B_S = \emptyset$, then $\mathcal{T}$ consists of a single leaf and the



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 3 (**i**) The ball b intersects all n segments and the segments have diameter larger than diam(b), so the density of the set of segments is n. (**ii**) Any ball b, no matter where it is placed or what its size is, intersects at most three triangles with diameter at least diam(b), so the density of the set of triangles is 3



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 4 Two cases in the construction of the BSP tree

recursion ends; otherwise we proceed as follows. Let σ_{NE}, σ_{SE}, σ_{SW}, and σ_{NW} denote the four quadrants of σ. We now have two cases, illustrated in Fig. 4.

Case (i): all points in $B_S(\sigma)$ lie in the same quadrant. Let σ' be the smallest square sharing a corner with σ and containing all points from $B_S(\sigma)$ in its interior or on its boundary. Split σ into three regions using a vertical and a horizontal splitting line such that σ' is one of those regions; see Fig. 4(i). Recursively construct a BSP tree for the square σ' with respect to the set $B_S(\sigma')$ of points lying in the interior of σ'.

Case (ii): not all points in $B_S(\sigma)$ lie in the same quadrant. Split σ into four quadrants using a vertical and two horizontal splitting lines; see Fig. 4(ii). Recursively construct a BSP tree for each quadrant with respect to the points lying in its interior.

The construction produces a subdivision of the initial square into $O(n)$ leaf regions, which are squares or rectangles and which do not contain points from B_S in their interior. Using Lemma 1, one can argue that each leaf region intersects $O(\lambda)$ objects.

Compressed Quadtrees

The construction of a compressed quadtree for a low-density set S is also based on the set B_S of bounding-box vertices: we construct a compressed quadtree for B_S, where we stop the recursive construction when a square contains bounding-box vertices from at most one object in S or when all bounding-box vertices inside the square coincide. Figure 2 illustrates the process. The resulting compressed quadtree has $O(n)$ leaf regions, which are canonical squares or donuts. Again using Lemma 1, one can argue that each leaf region intersects $O(\lambda)$ objects.

Improvements and Generalizations

The constructions above guarantee that each region in the space decomposition is intersected by $O(\lambda)$ objects and that the number of regions is $O(n)$. Hence, the total number of the object fragments is $O(\lambda n)$. The main idea behind the introduction of the density λ is that in practice λ is often a small constant. Nevertheless, it is (at least from a theoretical point of view) desirable to get rid of the dependency on λ in the number of fragments. This is possible by reducing the number of regions in the decomposition to (n/λ). To this end, we allow leaf regions to contain up to $O(\lambda)$ bounding-box vertices. Note that Lemma 1 implies that a square with $O(\lambda)$ bounding-box vertices inside intersects $O(\lambda)$ objects. If implemented correctly, this idea leads to decompositions with $O(n/\lambda)$ regions each of which intersects $O(\lambda)$ objects, both for binary space partitions [2, Section 12.5] and for compressed quadtrees [4]. The results can also be generalized to higher dimensions, giving the following theorem.

Theorem 1 *Let S be a set of n objects in $\mathbb{R}^d$ and let $\lambda :=$ density(S). There is a binary space partition for S consisting of $O(n/\lambda)$ leaf regions, each intersecting $O(\lambda)$ objects. Similarly, there is a compressed quadtree with $O(n/\lambda)$ leaf regions, each intersecting $O(\lambda)$ objects.*

Cross-References

▸ Binary Space Partitions
▸ Quadtrees and Morton Indexing

Recommended Reading

1. De Berg M (2000) Linear size binary space partitions for uncluttered scenes. Algorithmica 28(3):353–366
2. De Berg M, Cheong O, Van Kreveld M, Overmars M (2008) Computational geometry: algorithms and applications, 3rd edn. Springer, Berlin/Heidelberg
3. De Berg M, Katz M, Van der Stappen AF, Vleugels J (2002) Realistic input models for geometric algorithms. Algorithmica 34(1):81–97
4. De Berg M, Haverkort H, Thite S, Toma L (2010) Star-quadtrees and guard-quadtrees: I/O-efficient indexes for fat triangulations and low-density planar subdivisions. Comput Geom Theory Appl 43:493–513
5. Chazelle B (1984) Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. SIAM J Comput 13(3):488–507
6. Paterson MS and Yao FF (1990) Efficient binary space partitions for hidden-surface removal and solid modeling. Discret Comput Geom 5(5):485–503
7. Tóth CD (2003) A note on binary plane partitions. Discret Comput Geom 30(1):3–16
8. Tóth CD (2011) Binary plane partitions for disjoint line segments. Discret Comput Geom 45(4):617–646

High Performance Algorithm Engineering for Large-Scale Problems

David A. Bader
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords

Experimental algorithmics

Years and Authors of Summarized Original Work

2005; Bader

Problem Definition

Algorithm engineering refers to the process required to transform a pencil-and-paper algorithm into a robust, efficient, well tested, and easily usable implementation. Thus it encompasses a number of topics, from modeling cache behavior to the principles of good software engineering; its main focus, however, is experimentation. In that sense, it may be viewed as a recent outgrowth of *Experimental Algorithmics* [14], which is specifically devoted to the development of methods, tools, and practices for assessing and refining algorithms through experimentation. The *ACM Journal of Experimental Algorithmics (JEA)*, at URL www.jea.acm.org, is devoted to this area.

High-performance algorithm engineering [2] focuses on one of the many facets of algorithm engineering: speed. The high-performance aspect does not immediately imply parallelism; in fact, in any highly parallel task, most of the impact of high-performance algorithm engineering tends to come from refining the serial part of the code.

The term *algorithm engineering* was first used with specificity in 1997, with the organization of the first *Workshop on Algorithm Engineering (WAE 97)*. Since then, this workshop has taken place every summer in Europe. The 1998 *Workshop on Algorithms and Experiments (ALEX98)* was held in Italy and provided a discussion forum for researchers and practitioners interested in the design, analyzes and experimental testing of exact and heuristic algorithms. A sibling workshop was started in the Unites States in 1999, the *Workshop on Algorithm Engineering and Experiments (ALENEX99)*, which has taken place every winter, colocated with the *ACM/SIAM Symposium on Discrete Algorithms (SODA)*.

Key Results

Parallel computing has two closely related main uses. First, with more memory and storage resources than available on a single workstation, a parallel computer can solve correspondingly larger instances of the same problems. This increase in size can translate into running higher-fidelity simulations, handling higher volumes of information in data-intensive applications, and answering larger numbers of queries and datamining requests in corporate databases. Secondly, with more processors and larger aggregate memory subsystems than available on a single workstation, a parallel computer can often solve problems faster. This increase in speed can also translate into all of the advantages listed above, but perhaps its crucial advantage is in turnaround time. When the computation is part of a real-time system, such as weather forecasting, financial investment decision-making, or tracking and guidance systems, turnaround time is obviously the critical issue. A less obvious benefit of shortened turnaround time is higher-quality work: when a computational experiment takes less than an hour, the researcher can afford the luxury of exploration – running several different scenarios in order to gain a better understanding of the phenomena being studied.

In algorithm engineering, the aim is to present repeatable results through experiments that apply to a broader class of computers than the specific make of computer system used during the experiment. For sequential computing, empirical results are often fairly machine-independent. While machine characteristics such as word size, cache and main memory sizes, and processor and bus speeds differ, comparisons across different uniprocessor machines show the same trends. In particular, the number of memory accesses and processor operations remains fairly constant (or within a small constant factor). In high-performance algorithm engineering with parallel computers, on the other hand, this portability is usually absent: each machine and environment is its own special case. One obvious reason is major differences in hardware that affect the balance of communication and computation costs – a true shared-memory machine exhibits very different behavior from that of a cluster based on commodity networks.

Another reason is that the communication libraries and parallel programming environments (e.g., MPI [12], OpenMP [16], and High-Performance Fortran [10]), as well as the parallel algorithm packages (e.g., fast Fourier transforms using FFTW [6] or parallelized linear algebra routines in ScaLAPACK [4]), often exhibit differing performance on different types of parallel platforms. When multiple library packages exist for the same task, a user may observe different running times for each library version even on the same platform. Thus a running-time analysis should clearly separate the time spent in the user code from that spent in various library calls. Indeed, if particular library calls contribute significantly to the running time, the number of such calls and running time for each call should be recorded and used in the analysis, thereby helping library developers focus on the most cost-effective improvements. For example, in a simple message-passing program, one can characterize the work done by keeping track of sequential work, communication volume, and number of communications. A more general program using the collective communication routines of MPI could also count the number of calls to these routines. Several packages are available to instrument MPI codes in order to capture such data (e.g., MPICH's nupshot [8], Pablo [17], and Vampir [15]). The SKaMPI benchmark [18] allows running-time predictions based on such measurements even if the target machine is not available for program development. SKaMPI was designed for robustness, accuracy, portability, and efficiency; For example, SKaMPI adaptively controls how often measurements are repeated, adaptively refines message-length and step-width at "interesting" points, recovers from crashes, and automatically generates reports.

Applications

The following are several examples of algorithm engineering studies for high-performance and parallelcomputing.

1. Bader's prior publications (see [2] and http://www.cc.gatech.edu/~bader) contain many empirical studies of parallel algorithms for

combinatorial problems like sorting, selection, graph algorithms, and image processing.

2. In a recent demonstration of the power of high-performance algorithm engineering, a million-fold speed-up was achieved through a combination of a 2,000-fold speedup in the serial execution of the code and a 512-fold speedup due to parallelism (a speed-up, however, that will scale to any number of processors) [13]. (In a further demonstration of algorithm engineering, additional refinements in the search and bounding strategies have added another speedup to the serial part of about 1,000, for an overall speedup in excess of 2 billion)
3. JáJá and Helman conducted empirical studies for prefix computations, sorting, and list-ranking, on symmetric multiprocessors. The sorting research (see [9]) extends Vitter's external Parallel Disk Model to the internal memory hierarchy of SMPs and uses this new computational model to analyze a general-purpose sample sort that operates efficiently in shared-memory. The performance evaluation uses nine well-defined benchmarks. The benchmarks include input distributions commonly used for sorting benchmarks (such as keys selected uniformly and at random), but also benchmarks designed to challenge the implementation through load imbalance and memory contention and to circumvent algorithmic design choices based on specific input properties (such as data distribution, presence of duplicate keys, pre-sorted inputs, etc.).
4. In [3] Blelloch et al. compare through analysis and implementation three sorting algorithms on the Thinking Machines CM-2. Despite the use of an outdated (and no longer available) platform, this paper is a gem and should be required reading for every parallel algorithm designer. In one of the first studies of its kind, the authors estimate running times of four of the machine's primitives, then analyze the steps of the three sorting algorithms in terms of these parameters. The experimental studies of the performance are normalized to provide clear comparison of how the algorithms scale with input size on a 32K-processor CM-2.
5. Vitter et al. provide the canonical theoretic foundation for I/O-intensive experimental algorithmics using external parallel disks (e.g., see [1, 19, 20]). Examples from sorting, FFT, permuting, and matrix transposition problems are used to demonstrate the parallel disk model.
6. Juurlink and Wijshoff [11] perform one of the first detailed experimental accounts on the preciseness of several parallel computation models on five parallel platforms. The authors discuss the predictive capabilities of the models, compare the models to find out which allows for the design of the most efficient parallel algorithms, and experimentally compare the performance of algorithms designed with the model versus those designed with machine-specific characteristics in mind. The authors derive model parameters for each platform, analyses for a variety of algorithms (matrix multiplication, bitonic sort, sample sort, all-pairs shortest path), and detailed performance comparisons.
7. The LogP model of Culler et al. [5] provides a realistic model for designing parallel algorithms for message-passing platforms. Its use is demonstrated for a number of problems, including sorting.
8. Several research groups have performed extensive algorithm engineering for high-performance numerical computing. One of the most prominent efforts is that led by Dongarra for ScaLAPACK [4], a scalable linear algebra library for parallel computers. ScaLAPACK encapsulates much of the high-performance algorithm engineering with significant impact to its users who require efficient parallel versions of matrix–matrix linear algebra routines. New approaches for automatically tuning the sequential library (e.g., LAPACK) are now available as the ATLAS package [21].

Open Problems

All of the tools and techniques developed over the last several years for algorithm engineering are applicable to high-performance algorithm engineering. However, many of these tools need further refinement. For example, cache-efficient programming is a key to performance but it is not yet well understood, mainly because of complex machine-dependent issues like limited associativity, virtual address translation, and increasingly deep hierarchies of high-performance machines. A key question is whether one can find simple models as a basis for algorithm development. For example, cache-oblivious algorithms [7] are efficient at all levels of the memory hierarchy in theory, but so far only few work well in practice. As another example, profiling a running program offers serious challenges in a serial environment (any profiling tool affects the behavior of what is being observed), but these challenges pale in comparison with those arising in a parallel or distributed environment (for instance, measuring communication bottlenecks may require hardware assistance from the network switches or at least reprogramming them, which is sure to affect their behavior). Designing efficient and portable algorithms for commodity multicore and manycore processors is an open challenge.

Cross-References

- ▶ Analyzing Cache Misses
- ▶ Cache-Oblivious B-Tree
- ▶ Cache-Oblivious Model
- ▶ Cache-Oblivious Sorting
- ▶ Engineering Algorithms for Computational Biology
- ▶ Engineering Algorithms for Large Network Applications
- ▶ Engineering Geometric Algorithms
- ▶ Experimental Methods for Algorithm Analysis
- ▶ External Sorting and Permuting
- ▶ Implementation Challenge for Shortest Paths
- ▶ Implementation Challenge for TSP Heuristics
- ▶ I/O-Model
- ▶ Visualization Techniques for Algorithm Engineering

Recommended Reading

1. Aggarwal A, Vitter J (1988) The input/output complexity of sorting and related problems. Commun ACM 31:1116–1127
2. Bader DA, Moret BME, Sanders P (2002) Algorithm engineering for parallel computation. In: Fleischer R, Meineche-Schmidt E, Moret BME (eds) Experimental algorithmics. Lecture notes in computer science, vol 2547. Springer, Berlin, pp 1–23
3. Blelloch GE, Leiserson CE, Maggs BM, Plaxton CG, Smith SJ, Zagha M (1998) An experimental analysis of parallel sorting algorithms. Theory Comput Syst 31(2):135–167
4. Choi J, Dongarra JJ, Pozo R, Walker DW (1992) ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers. In: The 4th symposium on the frontiers of massively parallel computations. McLean, pp 120–127
5. Culler DE, Karp RM, Patterson DA, Sahay A, Schauser KE, Santos E, Subramonian R, von Eicken T (1993) LogP: towards a realistic model of parallel computation. In: 4th symposium on principles and practice of parallel programming. ACM SIGPLAN, pp 1–12
6. Frigo M, Johnson SG (1998) FFTW: an adaptive software architecture for the FFT. In: Proceedings of IEEE international conference on acoustics, speech, and signal processing, Seattle, vol 3, pp 1381–1384
7. Frigo M, Leiserson CE, Prokop H, Ramachandran, S (1999) Cacheoblivious algorithms. In: Proceedings of 40th annual symposium on foundations of computer science (FOCS-99), New York. IEEE, pp 285–297
8. Gropp W, Lusk E, Doss N, Skjellum A (1996) A high-performance, portable implementation of the MPI message passing interface standard. Technical report, Argonne National Laboratory, Argonne. www.mcs.anl.gov/mpi/mpich/
9. Helman DR, JáJá J (1998) Sorting on clusters of SMP's. In: Proceedings of 12th international parallel processing symposium, Orlando, pp 1–7
10. High Performance Fortran Forum (1993) High performance Fortran language specification, 1.0 edn., May 1993
11. Juurlink BHH, Wijshoff HAG (1998) A quantitative comparison of parallel computation models. ACM Trans Comput Syst 13(3):271–318
12. Message Passing Interface Forum (1995) MPI: a message-passing interface standard. Technical report, University of Tennessee, Knoxville, June 1995. Version 1.1
13. Moret BME, Bader DA, Warnow T (2002) High-performance algorithm engineering for computational phylogenetics. J Supercomput 22:99–111, Special issue on the best papers from ICCS'01
14. Moret BME, Shapiro HD (2001) Algorithms and experiments: the new (and old) methodology. J Univ Comput Sci 7(5):434–446
15. Nagel WE, Arnold A, Weber M, Hoppe HC, Solchenbach K (1996) VAMPIR: visualization and analysis of MPI resources. Supercomputer 63 12(1):69–80
16. OpenMP Architecture Review Board (1997) OpenMP: a proposed industry standard API for shared memory programming. www.openmp.org

17. Reed DA, Aydt RA, Noe RJ, Roth PC, Shields KA, Schwartz B, Tavera LF (1993) Scalable performance analysis: the Pablo performance analysis environment. In: Skjellum A (ed) Proceedings of scalable parallel libraries conference, Mississippi State University. IEEE Computer Society Press, pp 104–113
18. Reussner R, Sanders P, Träff J (1998, accepted) SKaMPI: a comprehensive benchmark for public benchmarking of MPI. Scientific programming, 2001. Conference version with Prechelt, L., Müller, M. In: Proceedings of EuroPVM/MPI
19. Vitter JS, Shriver EAM (1994) Algorithms for parallel memory. I: two-level memories. Algorithmica 12(2/3):110–147
20. Vitter JS, Shriver EAM (1994) Algorithms for parallel memory II: hierarchical multilevel memories. Algorithmica 12(2/3):148–169
21. Whaley R, Dongarra J (1998) Automatically tuned linear algebra software (ATLAS). In: Proceedings of supercomputing 98, Orlando. www.netlib.org/utk/people/JackDongarra/PAPERS/atlas-sc98.ps

Holant Problems

Jin-Yi Cai[1,2], Heng Guo[2], and Tyson Williams[2]
[1]Beijing University, Beijing, China
[2]Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA

Keywords

Computational complexity; Counting complexity; Holant problems; Partition functions

Years and Authors of Summarized Original Work

2011; Cai, Lu, Xia
2012; Huang, Lu
2013; Cai, Guo, Williams
2013; Guo, Lu, Valiant
2014; Cai, Guo, Williams

Problem Definition

The framework of Holant problems is intended to capture a class of sum-of-product computations in a more refined way than counting CSP problems and is inspired by Valiant's holographic algorithms [12] (also cf. entry ▶ Holographic Algorithms). A constraint function f, or *signature*, is a mapping from $[\kappa]^n$ to $\mathbb{C}$, representing a local contribution to a global sum. Here, $[\kappa]$ is a finite domain set, and n is the arity of f. The range is usually taken to be $\mathbb{C}$, but it can be replaced by any commutative semiring. A Holant problem Holant($\mathcal{F}$) is parameterized by a set of constraint functions $\mathcal{F}$. We usually focus on the Boolean domain, namely, $\kappa = 2$. For consideration of models of computation, we restrict function values to be complex algebraic numbers.

We allow multigraphs, namely, graphs with self-loops and parallel edges. A *signature grid* $\Omega = (G, \pi)$ of Holant($\mathcal{F}$) consists of a graph $G = (V, E)$, where π assigns each vertex $v \in V$ and its incident edges with some $f_v \in \mathcal{F}$ and its input variables. We say Ω is a *planar signature grid* if G is planar. The Holant problem on instance Ω is to evaluate

$$\text{Holant}(\Omega; \mathcal{F}) = \sum_{\sigma} \prod_{v \in V} f_v(\sigma \mid_{E(v)}),$$

a sum over all edge labelings $\sigma : E \to [\kappa]$, where $E(v)$ denotes the incident edges of v and $\sigma \mid_{E(v)}$ denotes the restriction of σ to $E(v)$. This is also known as the partition function in the statistical physics literature.

Formally, a set of signatures $\mathcal{F}$ defines the following Holant problem:

Name Holant($\mathcal{F}$)
Instance A *signature grid* $\Omega = (G, \pi)$
Output Holant($\Omega; \mathcal{F}$)

The problem Pl-Holant($\mathcal{F}$) is defined similarly using a planar signature grid.

A function f_v can be represented by listing its values in lexicographical order as in a truth table, which is a vector in $\mathbb{C}^{\kappa^{\deg(v)}}$ or as a tensor in $(\mathbb{C}^{\kappa})^{\otimes \deg(v)}$. Special focus has been put on *symmetric* signatures, which are functions invariant under any permutation of the input. An example is the EQUALITY signature $=_n$ of arity n. A Boolean symmetric function f of arity n can be listed as $[f_0, f_1, \ldots, f_n]$, where f_w is the function value of f when the input has Hamming weight w. Using this notation, an EQUALITY signature is $[1, 0, \dots, 0, 1]$. Another example is the EXACTONE signature $[0, 1, 0, \dots, 0]$. Clearly, the Holant problem defined by this signature counts the number of perfect matchings.

The set $\mathcal{F}$ is allowed to be an infinite set. For Holant($\mathcal{F}$) to be tractable, the problem must be computable in polynomial time even when the description of the signatures in the input Ω is included in the input size. In contrast, we say Holant($\mathcal{F}$) is #P-hard if there exists a finite subset of $\mathcal{F}$ for which the problem is #P-hard.

The Holant framework is a generalization and refinement of both counting graph homomorphisms and counting constraint satisfaction problems (see entry ▸ Complexity Dichotomies for Counting Graph Homomorphisms for more details and results).

Key Results

The Holant problem was introduced by Cai, Lu, and Xia [3], which also contains a dichotomy of Holant* for symmetric Boolean complex functions. The notation Holant* means that all unary functions are assumed to be available. This restriction is later weakened to only allow two constant functions that pin a variable to 0 or 1. This framework is called Holant^c. In [5], a dichotomy of Holant^c is obtained. The need to assume some freely available functions is finally avoided in [10]. In this paper, Huang and Lu proved a dichotomy for Holant but with the caveat that the functions must be real weighted. This result was later improved by Cai, Guo, and Williams [6], who proved a dichotomy for Holant parameterized by any set of symmetric Boolean complex functions.

We will give some necessary definitions and then state the dichotomy from [6]. First are several tractable families of functions over the Boolean domain.

Definition 1 A signature f of arity n is *degenerate* if there exist unary signatures $u_j \in \mathbb{C}^2$ $(1 \leq j \leq n)$ such that $f = u_1 \otimes \cdots \otimes u_n$. A symmetric degenerate signature has the form $u^{\otimes n}$.

Definition 2 A k-ary function $f(x_1, \dots, x_k)$ is of *affine* type if it has the form

$$\lambda \chi_{Ax=0} \cdot \sqrt{-1}^{\sum_{j=1}^{n} \langle \alpha_j, x \rangle},$$

where $\lambda \in \mathbb{C}$, $x = (x_1, x_2, \dots, x_k, 1)^{\mathrm{T}}$, A is a matrix over $\mathbb{F}_2$, α_j is a vector over $\mathbb{F}_2$, and χ is a 0–1 indicator function such that $\chi_{Ax=0}$ is 1 iff $Ax = 0$. Note that the dot product $\langle \alpha_j, x \rangle$ is calculated over $\mathbb{F}_2$, while the summation $\sum_{j=1}^{n}$ on the exponent of $i = \sqrt{-1}$ is evaluated as a sum mod 4 of 0–1 terms. We use $\mathscr{A}$ to denote the set of all affine-type functions.

An alternative but equivalent form for an affine-type function is $\lambda \chi_{Ax=0} \cdot \sqrt{-1}^{Q(x_1, x_2, \dots, x_k)}$ where $Q(\cdot)$ is a quadratic form with integer coefficients that are even for every cross term.

Definition 3 A function is of *product type* if it can be expressed as a product of unary functions, binary equality functions ($[1, 0, 1]$), and binary disequality functions ($[0, 1, 0]$), each applied to some of its variables. We use $\mathscr{P}$ to denote the set of product-type functions.

Definition 4 A function f is called *vanishing* if the value Holant($\Omega; \{f\}$) is 0 for every signature grid Ω. We use $\mathscr{V}$ to denote the set of vanishing functions.

For vanishing signatures, we need some more definitions.

Definition 5 An arity n symmetric signature of the form $f = [f_0, f_1, \dots, f_n]$ is in $\mathscr{R}_t^+$ for a nonnegative integer $t \geq 0$ if $t > n$ or for any $0 \leq k \leq n - t$, $f_k, \dots, f_{k+t}$ satisfy the recurrence relation

$$\binom{t}{t} i^t f_{k+t} + \binom{t}{t-1} i^{t-1} f_{k+t-1} + \cdots + \binom{t}{0} i^0 f_k = 0. \quad (1)$$

H

We define $\mathscr{R}_t^-$ similarly but with $-i$ in place of i in (1).

With $\mathscr{R}_t^\pm$, one can define the recurrence degree of a function f.

Definition 6 For a nonzero symmetric signature f of arity n, it is of *positive* (resp. *negative*) *recurrence degree* $t \leq n$, denoted by $\mathrm{rd}^+(f) = t$ (resp. $\mathrm{rd}^-(f) = t$), if and only if $f \in \mathscr{R}_{t+1}^+ - \mathscr{R}_t^+$ (resp. $f \in \mathscr{R}_{t+1}^- - \mathscr{R}_t^-$). If f is the all-zero signature, we define $\mathrm{rd}^+(f) = \mathrm{rd}^-(f) = -1$.

In [6], it is shown that $f \in \mathscr{V}$ if and only if for either $\sigma = +$ or $-$, we have $2\mathrm{rd}^\sigma(f) < \mathrm{arity}(f)$. Accordingly, we split the set $\mathscr{V}$ of vanishing signatures in two.

Definition 7 We define $\mathscr{V}^\sigma$ for $\sigma \in \{+,-\}$ as

$$\mathscr{V}^\sigma = \{f \mid 2\mathrm{rd}^\sigma(f) < \mathrm{arity}(f)\}.$$

To state the dichotomy, we also need the notion of $\mathcal{F}$-transformable. For a matrix $T \in \mathbb{C}^{2\times 2}$, and a signature set $\mathcal{F}$, define $T\mathcal{F} = \{g \mid \exists f \in \mathcal{F}$ of arity n, $g = T^{\otimes n} f\}$. Here, we view the signatures as column vectors. Let $=_2$ be the equality function of arity 2.

Definition 8 A signature set $\mathcal{F}'$ is $\mathcal{F}$-transformable if there exists a non-singular matrix $T \in \mathbb{C}^{2\times 2}$ such that $\mathcal{F}' \subseteq T\mathcal{F}$ and $(=_2)T^{\otimes 2} \in \mathcal{F}$.

If a set of functions $\mathcal{F}'$ is $\mathcal{F}$-transformable and $\mathcal{F}$ is a tractable set, then Holant($\mathcal{F}'$) is tractable as well.

The dichotomy of Holant problems over symmetric Boolean complex functions is stated as follows.

Theorem 1 ([6]) *Let $\mathcal{F}$ be any set of symmetric, complex-valued signatures in Boolean variables. Then,* Holant($\mathcal{F}$) *is #P-hard unless $\mathcal{F}$ satisfies one of the following conditions, in which case the problem is in P:*

1. *All nondegenerate signatures in $\mathcal{F}$ are of arity at most* 2;
2. *$\mathcal{F}$ is $\mathscr{A}$-transformable;*
3. *$\mathcal{F}$ is $\mathscr{P}$-transformable;*
4. *$\mathcal{F} \subseteq \mathscr{V}^\sigma \cup \{f \in \mathscr{R}_2^\sigma \mid \mathrm{arity}(f) = 2\}$ for some $\sigma \in \{+,-\}$;*
5. *All nondegenerate signatures in $\mathcal{F}$ are in $\mathscr{R}_2^\sigma$ for some $\sigma \in \{+,-\}$.*

Theorem 1 is about Holant problems parameterized by symmetric Boolean complex functions over general graphs. Holant problems are studied in other settings as well. For planar graphs, [2] contains a dichotomy for Holant^c with real symmetric functions. There are signature sets that are #P-hard over general graphs but tractable over planar graphs. The algorithms for such sets are due to Valiant's holographic algorithms and the theory of matchgates [1, 12].

Another generalization looks at a broader range of functions. One may consider asymmetric functions as in [4], which contains a dichotomy for Holant* problems defined by asymmetric Boolean complex functions. One can also consider functions of larger domain size. For domain size 3, [7] contains a dichotomy for a single arity 3 symmetric complex function in the Holant* setting. For any constant domain size, [8] contains a dichotomy for a single arity 3 complex weighted function that satisfies a strong symmetry property.

One can consider constraint functions with a range other than $\mathbb{C}$. Replacing $\mathbb{C}$ by some finite field $\mathbb{F}_p$ for some prime p defines counting problems modulo p. The case $p = 2$ is called parity Holant problems. It is of special interest because computing the permanent modulo 2 is tractable, which implies a family of tractable matchgate functions even over general graphs. For parity Holant problems, a complete dichotomy for symmetric functions is obtained by Guo, Lu, and Valiant [9].

Open Problems

Unlike the progress in the general graph setting, the strongest known dichotomy results for planar Holant problems are rather limited. These planar dichotomies showed that newly tractable problems over planar graphs are captured by holographic algorithms with matchgates, but with restrictions like symmetric functions or regular graphs. The theory of holographic algorithms with matchgates can be applied to planar graphs and asymmetric signatures. A true test of its power would be to obtain an asymmetric complex weighted dichotomy of planar Holant problems. The situation is similarly limited for higher domain sizes, where things seem considerably more complicated. A reasonable first step in this direction would be to consider some restricted (yet still powerful) family of functions.

Despite the success for $\mathbb{F}_2$, little is known about the complexity of Holant problems over other finite fields or semirings. As Valiant showed in [11], counting problems modulo some finite modulus include some interesting and surprising phenomena. It deserves further research.

Cross-References

▸ Complexity Dichotomies for Counting Graph Homomorphisms
▸ Holographic Algorithms

Recommended Reading

1. Cai JY, Lu P (2011) Holographic algorithms: from art to science. J Comput Syst Sci 77(1):41–61
2. Cai JY, Lu P, Xia M (2010) Holographic algorithms with matchgates capture precisely tractable planar #CSP. In: FOCS, Las Vegas. IEEE Computer Society, pp 427–436
3. Cai JY, Lu P, Xia M (2011) Computational complexity of Holant problems. SIAM J Comput 40(4):1101–1132
4. Cai JY, Lu P, Xia M (2011) Dichotomy for Holant* problems of Boolean domain. In: SODA, San Francisco. SIAM, pp 1714–1728
5. Cai JY, Huang S, Lu P (2012) From Holant to #CSP and back: dichotomy for Holant^c problems. Algorithmica 64(3):511–533
6. Cai JY, Guo H, Williams T (2013) A complete dichotomy rises from the capture of vanishing signatures (extended abstract). In: STOC, Palo Alto. ACM, pp 635–644
7. Cai JY, Lu P, Xia M (2013) Dichotomy for Holant* problems with domain size 3. In: SODA, New Orleans. SIAM, pp 1278–1295
8. Cai JY, Guo H, Williams T (2014) The complexity of counting edge colorings and a dichotomy for some higher domain Holant problems. In: FOCS, Philadelphia. IEEE, pp 601–610
9. Guo H, Lu P, Valiant LG (2013) The complexity of symmetric Boolean parity Holant problems. SIAM J Comput 42(1):324–356
10. Huang S, Lu P (2012) A dichotomy for real weighted Holant problems. In: CCC, Porto. IEEE Computer Society, pp 96–106
11. Valiant LG (2006) Accidental algorithms. In: FOCS, Berkeley. IEEE, pp 509–517
12. Valiant LG (2008) Holographic algorithms. SIAM J Comput 37(5):1565–1594

Holographic Algorithms

Jin-Yi Cai[1,2], Pinyan Lu[3], and Mingji Xia[4]
[1]Beijing University, Beijing, China
[2]Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA
[3]Microsoft Research Asia, Shanghai, China
[4]The State Key Laboratory of Computer Science, Chinese Academy of Sciences, Beijing, China

Keywords

Bases; Counting problems; Holographic algorithms; Perfect matchings; Planar graphs

Years and Authors of Summarized Original Work

2006, 2008; Valiant
2008, 2009, 2010, 2011; Cai, Lu
2009; Cai, Choudhary, Lu
2014; Cai, Gorenstein

Problem Definition

Holographic algorithm, introduced by L. Valiant [11], is an algorithm design technique rather than a single algorithm for a particular problem. In essence, these algorithms are reductions to the FKT algorithm [7–9] to count the number of perfect matchings in a planar graph in polynomial time. Computation in these algorithms is expressed and interpreted through a choice of linear basis vectors in an exponential “holographic” mix, and then it is carried out by the FKT method via the Holant Theorem. This methodology has produced polynomial time algorithms for a variety of problems ranging from restrictive versions of satisfiability, vertex cover, to other graph problems such as edge orientation and node/edge deletion. No polynomial time algorithms were known for these problems, and some minor variations are known to be NP-hard (or even #P-hard).

Let $G = (V, E, W)$ be a weighted undirected planar graph, where V, E, and W are sets of vertices, edges, and edge weights, respectively. A matchgate is a tuple (G, X) where $X \subseteq V$ is a set of external nodes on the outer face. A matchgate is considered a generator or a recognizer matchgate when the external nodes are considered output or input nodes, respectively. They differ mainly in the way they are transformed. The external nodes are ordered clockwise on the external face. Γ is called an odd (resp. even) matchgate if it has an odd (resp. even) number of nodes.

Each matchgate is assigned a *signature* tensor. A generator Γ with m output nodes is assigned a contravariant tensor $\mathbf{G} \in V_0^m$ of type $\binom{m}{0}$, where V_0^m is the tensor space spanned by the m-fold tensor products of the standard basis $\mathbf{b} = [\mathbf{b}_0, \mathbf{b}_1] = \left[\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}\right]$. The tensor $\mathbf{G}$ under the standard basis $\mathbf{b}$ has the form

$$\sum G^{i_1 i_2 \dots i_m} \mathbf{b}_{i_1} \otimes \mathbf{b}_{i_2} \otimes \cdots \otimes \mathbf{b}_{i_m},$$

where

$$G^{i_1 i_2 \dots i_m} = \mathrm{PerfMatch}(G - Z).$$

Here Z is the subset of the output nodes of Γ having the characteristic sequence $\chi_Z = i_1 i_2 \dots i_m \in \{0,1\}^m$, $\mathrm{PerfMatch}(G - Z) = \sum_M \prod_{(i,j) \in M} w_{ij}$ is a sum over all perfect matchings M in the graph $G - Z$ obtained from G by removing Z and its incident edges, and w_{ij} is the weight of the edge (i, j). Similarly a recognizer Γ' with underlying graph G' having m input nodes is assigned a covariant tensor $\mathbf{R} \in V_m^0$ of type $\binom{0}{m}$. This tensor under the standard (dual) basis $\mathbf{b}^*$ has the form

$$\sum R_{i_1 i_2 \dots i_m} \mathbf{b}^{i_1} \otimes \mathbf{b}^{i_2} \otimes \cdots \otimes \mathbf{b}^{i_m},$$

where

$$R_{i_1 i_2 \dots i_m} = \mathrm{PerfMatch}(G' - Z),$$

and Z is the subset of the input nodes of Γ' having the characteristic sequence $\chi_Z = i_1 i_2 \dots i_m$.

As a contravariant tensor, $\mathbf{G}$ transforms as follows. Under a basis transformation $\boldsymbol{\beta}_j = \sum_i \mathbf{b}_i t_j^i$,

$$(G')^{j_1 j_2 \dots j_m} = \sum G^{i_1 i_2 \dots i_m} \tilde{t}_{i_1}^{j_1} \tilde{t}_{i_2}^{j_2} \dots \tilde{t}_{i_m}^{j_m},$$

where $(\tilde{t}_i^j)$ is the inverse matrix of (t_j^i). Similarly, $\mathbf{R}$ transforms as a covariant tensor, namely,

$$(R')_{j_1 j_2 \dots j_m} = \sum R_{i_1 i_2 \dots i_m} t_{j_1}^{i_1} t_{j_2}^{i_2} \dots t_{j_m}^{i_m}.$$

A signature is *symmetric* if each entry only depends on the Hamming weight of the index $i_1 i_2 \dots i_m$. This notion is invariant under a basis transformation. A symmetric signature is denoted by $[\sigma_0, \sigma_1, \dots, \sigma_m]$, where σ_i denotes the value of a signature entry whose Hamming weight of its index is i.

A *matchgrid* $\Omega = (A, B, C)$ is a weighted planar graph consisting of a disjoint union of: a set of g generators $A = (A_1, \dots, A_g)$, a set of r recognizers $B = (B_1, \dots, B_r)$, and a set of f connecting edges $C = (C_1, \dots, C_f)$, where each C_i edge has weight 1 and joins an output node of a generator with an input node of a recognizer, so that every input and output node in every constituent matchgate has exactly one such incident connecting edge.

Let $\mathbf{G} = \bigotimes_{i=1}^{g} \mathbf{G}(A_i)$ be the tensor product of all the generator signatures, and let $\mathbf{R} = \bigotimes_{j=1}^{r} \mathbf{R}(B_j)$ be the tensor product of all the recognizer signatures. Then Holant_Ω is defined to be the contraction of the two product tensors, under some basis $\boldsymbol{\beta}$, where the corresponding indices match up according to the f connecting edges C_k:

$$\mathrm{Holant}_\Omega = \langle \mathbf{R}, \mathbf{G} \rangle = \sum_{x \in \boldsymbol{\beta}^{\otimes f}} \left\{ [\Pi_{1 \le i \le g} \mathbf{G}(A_i, x|_{A_i})] \cdot [\Pi_{1 \le j \le r} \mathbf{R}(B_j, x^*|_{B_j})] \right\}. \quad (1)$$

If we write the covariant tensor $\mathbf{R}$ as a row vector of dimension 2^f, write the contravariant tensor $\mathbf{G}$ as a column vector of dimension 2^f, both indexed by some common ordering of the connecting edges, then Holant_Ω is just the dot product of these two vectors. Valiant's beautiful Holant Theorem is as follows:

Theorem 1 (Valiant) *For any matchgrid Ω over any basis $\boldsymbol{\beta}$, let G be its underlying weighted graph, then*

$$\mathrm{Holant}_\Omega = \mathrm{PerfMatch}(G).$$

The FKT algorithm can compute the perfect matching polynomial $\mathrm{PerfMatch}(G)$ for a planar graph in polynomial time. This gives a polynomial time algorithm to compute Holant_Ω.

Key Results

To design a holographic algorithm for a given problem, the creative part is to formalize the given problem as a Holant problem. The theory of holographic algorithms is trying to answer the second question: given a Holant problem, can we find a basis transformation so that all the signatures in the Holant problem can be realized by some matchgates on that basis? More formally, we want to solve the following simultaneous realizability problem (SRP).

Definition 1 Simultaneous Realizability Problem (SRP):

Input: A set of constraint functions for generators and recognizers.

Output: A common basis under which these functions can be simultaneously realized by matchgate signatures, if any exists; "NO" if they are not simultaneously realizable.

The theory of matchgates and holographic algorithms provides a systematic understanding of which constraint functions can be realized by matchgates, the structure for the bases, and finally solve the simultaneous realizability problem.

Matchgate Identities

There is a set of algebraic identities [1, 6] which completely characterizes signatures directly realizable without basis transformation by matchgates for any number of inputs and outputs. These identities are derived from Grassmann-Plücker identities for Pfaffians.

Patterns α, β are m-bit strings, i.e., $\alpha, \beta \in \{0,1\}^m$. A position vector $P = \{p_i\}, i \in [l]$ is a subsequence of $\{1, 2, \ldots, m\}$, i.e., $p_i \in [m]$ and $p_1 < p_2 < \cdots < p_l$. We also use p to denote the m-bit string, whose $(p_1, p_2, \ldots, p_l)$-th bits are 1 and others are 0. Let $e_i \in \{0,1\}^m$ be the pattern with 1 in the i-th bit and 0 elsewhere. Let $\alpha, \beta \in \{0,1\}^m$ be any pattern, and let $P = \{p_i\} = \alpha + \beta$, $i \in [l]$ be their bit-wise XOR as a position vector. Then, we have the following identity:

$$\sum_{i=1}^{l} (-1)^i G^{\alpha + e_{p_i}} G^{\beta + e_{p_i}} = 0. \quad (2)$$

A tensor $\mathbf{G} = (G^{i_1, \ldots, i_m})$ is realizable as the signature, without basis transformation, of some planar matchgate iff it satisfies the matchgate identities (2) for all α and β.

Basis Collapse

When we consider basis transformations for holographic algorithms, we mainly focus on invertible transformations, and these are bases of dimension 2. However, in a paper called "accidental algorithm" [10], Valiant showed that a basis of dimension 4 can be used to solve in P an interesting (restrictive SAT) counting problem mod 7. In a later paper [4], we have shown, among other things, that for this particular problem, this use of bases of size 2 is unnecessary. Then, in a sequence of two papers [2, 3], we completely resolve the problem of the power of higher dimensional bases. We prove that 2-dimensional bases are universal for holographic algorithms in the Boolean domain.

Theorem 2 (Basis Collapse Theorem) *Any holographic algorithm on a basis of any dimension which employs at least one nondegenerate generator can be efficiently transformed to a holographic algorithm in a basis of dimension 2. More precisely, if generators $G_1, G_2, \ldots, G_s$ and recognizers $R_1, R_2, \ldots, R_t$ are simultaneously realizable on a basis T of any dimension, and not all generators are degenerate, then all the generators and recognizers are simultaneously realizable in a basis $\hat{T}$ of dimension 2.*

From Art to Science

Based on the characterization for matchgate signatures and basis transformations, we can solve the simultaneous realizability problem [5]. In order to investigate the realizability of signatures, it is useful to introduce a basis manifold $\mathcal{M}$, which is defined to be the set of all possible bases modulo an equivalence relation. One can characterize in terms of $\mathcal{M}$ all realizable symmetric signatures under basis transformations. This structural understanding gives: (i) a uniform account of all the previous successes of holographic algorithms using symmetric signatures [10, 11]; (ii) generalizations to solve other problems, when this is possible; and (iii) a proof when this is not possible.

Applications

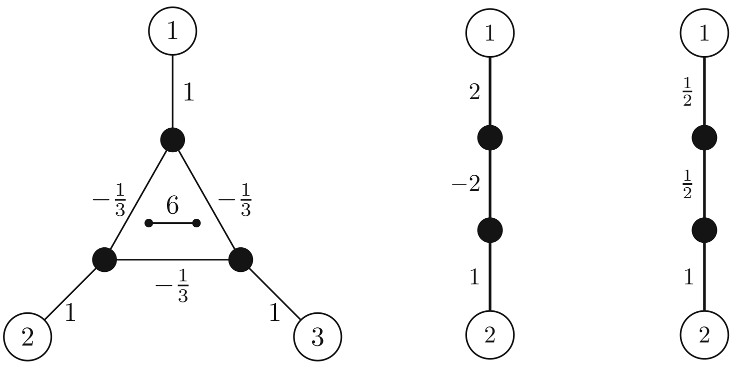
In this section, we list a few problems which can be solved by holographic algorithms.

#PL-3-NAE-ICE

INPUT: A planar graph $G = (V, E)$ of maximum degree 3.

OUTPUT: The number of orientations such that no node has all incident edges directed toward it or all incident edges directed away from it.

Hence, #PL-3-NAE-ICE counts the number of no-sink-no-source orientations. A node of degree one will preclude such an orientation. We assume every node has degree 2 or 3. To solve this problem by a holographic algorithm with matchgates, we design a signature grid based on G as follows: We attach to each node of degree 3 a generator with signature $[0, 1, 1, 0]$. This represents a NOT-ALL-EQUAL or NAE gate of arity 3. For any node of degree 2, we use a generator with the binary NAE (i.e., a binary DISEQUALITY) signature $(\neq_2) = [0, 1, 0]$. For each edge in E, we use a recognizer with signature $(\neq_2)$, which stands for an orientation from one node to the other. (To express such a problem, it is completely arbitrary



Holographic Algorithms, Fig. 1 Some matchgates used in #PL-3-NAE-ICE to label one side as generators and the other side as recognizers.) From the given planar graph G, we obtain a signature grid Ω, where the underlying graph G' is the edge-vertex incidence graph of G. By definition, Holant_Ω is an exponential sum where each term is a product of appropriate entries of the signatures. Each term is indexed by a 0–1 assignment on all edges of G'; it has a value of 0 or 1, and it has a value of 1 iff it corresponds to an orientation of G such that at every vertex of G the local NAE constraint is satisfied. Therefore, Holant_Ω is precisely the number of valid orientations required by #PL-3-NAE-ICE.

Note that the signature $[0, 1, 1, 0]$ is not the signature of any matchgate. A simple reason for this is that a matchgate signature, being defined in terms of perfect matchings, cannot have nonzero values for inputs of both odd and even Hamming weights.

However, under a holographic transformation using $H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$,

$$H^{\otimes 3}[0, 1, 1, 0] = H^{\otimes 3}\left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix}^{\otimes 3} - \begin{bmatrix} 1 \\ 0 \end{bmatrix}^{\otimes 3} - \begin{bmatrix} 0 \\ 1 \end{bmatrix}^{\otimes 3} \right\} = [6, 0, -2, 0],$$

$$H^{\otimes 2}[0, 1, 0] = H^{\otimes 2}\left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix}^{\otimes 2} - \begin{bmatrix} 1 \\ 0 \end{bmatrix}^{\otimes 2} - \begin{bmatrix} 0 \\ 1 \end{bmatrix}^{\otimes 2} \right\} = [2, 0, -2],$$

and

$$[0, 1, 0](H^{-1})^{\otimes 2} = \frac{1}{2}[1, 0, -1].$$

These signatures are all realizable as matchgate signatures by verifying all the matchgate identities. More concretely, we can exhibit the requisite three matchgates in Fig. 1.

Hence, #PL-3-NAE-ICE is precisely the following Holant problem on planar graphs:

$$\begin{aligned} &\text{Holant}([0, 1, 0] \mid [0, 1, 0], [0, 1, 1, 0]) \\ &\equiv_T \text{Holant}(\tfrac{1}{2}[1, 0, -1] \mid [2, 0, -2], [6, 0, -2, 0]). \end{aligned}$$

Now we may replace each signature $\frac{1}{2}[1, 0, -1]$, $[2, 0, -2]$, and $[6, 0, -2, 0]$ in Ω by their corresponding matchgates, and then we can compute Holant_Ω in polynomial time by Kasteleyn's algorithm.

The next problem is a satisfiability problem.

#PL-3-NAE-SAT

INPUT: A planar formula Φ consisting of a conjunction of NAE clauses each of size 3.

OUTPUT: The number of satisfying assignments of Φ.

This is a variant of 3SAT. A Boolean formula is planar if it can be represented by a planar graph where vertices represent variables and clauses, and there is an edge iff the variable or its negation appears in that clause. The SAT problem is when the gate for each clause is the Boolean OR. When SAT is restricted to planar formulae, it is still NP-complete, and its corresponding counting problem is #P-complete. Moreover, for many connectives other than NAE (e.g., EXACTLY ONE), the unrestricted or the planar decision problems are still NP-complete, and the corresponding counting problems are #P-complete.

We design a signature grid as follows: To each NAE clause, we assign a generator with signature $[0, 1, 1, 0]$. To each Boolean variable, we assign a generator with signature $(=_k)$ where k is the number of clauses the variable appears, either negated or unnegated. Further, if a variable occurrence is negated, we have a recognizer $[0, 1, 0]$ along the edge that joins the variable generator and the NAE generator, and if the variable occurrence is unnegated, then we use a recognizer $[1, 0, 1]$ instead. Under a holographic transformation using H, $(=_k)$ is transformed to

$$H^{\otimes k}\left\{\begin{bmatrix}1\\0\end{bmatrix}^{\otimes k}+\begin{bmatrix}0\\1\end{bmatrix}^{\otimes k}\right\}=\begin{bmatrix}1\\1\end{bmatrix}^{\otimes k}+\begin{bmatrix}1\\-1\end{bmatrix}^{\otimes k}$$
$$=2[1,0,1,0,\ldots].$$

It can be verified that all the signatures used satisfy all matchgate identities and thus can be realized by matchgates under the holographic transformation.

Recommended Reading

1. Cai JY, Gorenstein A (2014) Matchgates revisited. Theory Comput 10(7):167–197
2. Cai JY, Lu P (2008) Basis collapse in holographic algorithms. Comput Complex 17(2):254–281
3. Cai JY, Lu P (2009) Holographic algorithms: the power of dimensionality resolved. Theor Comput Sci Comput Sci 410(18):1618–1628
4. Cai JY, Lu P (2010) On symmetric signatures in holographic algorithms. Theory Comput Syst 46(3):398–415
5. Cai JY, Lu P (2011) Holographic algorithms: from art to science. J Comput Syst Sci 77(1):41–61
6. Cai JY, Choudhary V, Lu P (2009) On the theory of matchgate computations. Theory Comput Syst 45(1):108–132
7. Kasteleyn PW (1961) The statistics of dimers on a lattice. Physica 27:1209–1225
8. Kasteleyn PW (1967) Graph theory and crystal physics. In: Harary F (ed) Graph theory and theoretical physics. Academic, London, pp 43–110
9. Temperley HNV, Fisher ME (1961) Dimer problem in statistical mechanics – an exact result. Philos Mag 6:1061–1063
10. Valiant LG (2006) Accidental algorthims. In: FOCS '06: proceedings of the 47th annual IEEE symposium on foundations of computer science. IEEE Computer Society, Washington, pp 509–517. doi:http://dx.doi.org/10.1109/FOCS.2006.7
11. Valiant LG (2008) Holographic algorithms. SIAM J Comput 37(5):1565–1594. doi:http://dx.doi.org/10.1137/070682575

Hospitals/Residents Problem

David F. Manlove
School of Computing Science, University of Glasgow, Glasgow, UK

Keywords

Matching; Stability

Synonyms

College admissions problem; Stable admissions problem; Stable assignment problem; Stable b-matching problem; University admissions problem

Years and Authors of Summarized Original Work

1962; Gale, Shapley

Problem Definition

An instance I of the *Hospitals/Residents problem* (HR) [6, 7, 18] involves a set $R = \{r_1, \ldots, r_n\}$ of *residents* and a set $H = \{h_1, \ldots, h_m\}$ of *hospitals*. Each hospital $h_j \in H$ has a positive integral *capacity*, denoted by c_j. Also, each resident $r_i \in R$ has a *preference list* in which he ranks in strict order a subset of H. A pair $(r_i, h_j) \in R \times H$ is said to be *acceptable* if h_j appears in r_i's preference list; in this case r_i is said to *find* h_j *acceptable*. Similarly each hospital $h_j \in H$ has a preference list in which it ranks in strict order those residents who find h_j acceptable. Given any three agents $x, y, z \in R \cup H$, x is said to *prefer* y to z if x finds each of y and z acceptable, and y precedes z on x's preference list. Let $C = \sum_{h_j \in H} c_j$.

Let A denote the set of acceptable pairs in I, and let $L = |A|$. An *assignment* M is a subset of A. If $(r_i, h_j) \in M$, r_i is said to be *assigned to* h_j, and h_j is *assigned* r_i. For each $q \in R \cup H$, the set of assignees of q in M is denoted by $M(q)$. If $r_i \in R$ and $M(r_i) = \emptyset$, r_i is said to be *unassigned*; otherwise r_i is *assigned*. Similarly, any hospital $h_j \in H$ is *under-subscribed*, *full*, or *over-subscribed* according as $|M(h_j)|$ is less than, equal to, or greater than c_j, respectively.

A *matching* M is an assignment such that $|M(r_i)| \le 1$ for each $r_i \in R$ and $|M(h_j)| \le c_j$ for each $h_j \in H$ (i.e., no resident is assigned to an unacceptable hospital, each resident is assigned to at most one hospital, and no hospital is over-subscribed). For notational convenience, given a matching M and a resident $r_i \in R$ such that $M(r_i) \neq \emptyset$, where there is no ambiguity, the notation $M(r_i)$ is also used to refer to the single member of $M(r_i)$.

A pair $(r_i, h_j) \in A \backslash M$ *blocks* a matching M or is a *blocking pair* for M, if the following conditions are satisfied relative to M:

1. r_i is unassigned or prefers h_j to $M(r_i)$;
2. h_j is under-subscribed or prefers r_i to at least one member of $M(h_j)$ (or both).

A matching M is said to be *stable* if it admits no blocking pair. Given an instance I of HR, the problem is to find a stable matching in I.

Key Results

HR was first defined by Gale and Shapley [6] under the name "College Admissions Problem." In their seminal paper, the authors' primary consideration is the classical *Stable Marriage problem* (SM; see Entries ▸ Stable Marriage and ▸ Optimal Stable Marriage), which is a special case of HR in which $n = m$, $A = R \times H$, and $c_j = 1$ for all $h_j \in H$ – in this case, the residents and hospitals are more commonly referred to as the *men* and *women*, respectively. Gale and Shapley showed that every instance I of HR admits at least one stable matching. Their proof of this result is constructive, i.e., an algorithm for finding a stable matching in I is described. This algorithm has become known as the *Gale/Shapley algorithm.*

An extended version of the Gale/Shapley algorithm for HR is shown in Fig. 1. The algorithm involves a sequence of *apply* and *delete* operations. At each iteration of the while loop, some unassigned resident r_i with a nonempty preference list applies to the first hospital h_j on his list and becomes provisionally assigned to h_j (this assignment could subsequently be broken). If h_j becomes over-subscribed as a result of this assignment, then h_j rejects its worst assigned resident r_k. Next, if h_j is full (irrespective of whether h_j was over-subscribed earlier in the same loop iteration), then for each resident r_l that h_j finds less desirable than its worst assigned resident r_k, the algorithm *deletes the pair* (r_l, h_j), which comprises deleting h_j from r_l's preference list and vice versa.

Given that the above algorithm involves residents applying to hospitals, it has become known as the *Resident-oriented* Gale/Shapley algorithm, or RGS algorithm for short [7, Section 1.6.3]. The RGS algorithm terminates with a stable matching, given an instance of HR [6] [7, Theorem 1.6.2]. Using a suitable choice of data structures (extending those described in [7, Section 1.2.3]), the RGS algorithm can be implemented to run in $O(L)$ time. This algorithm produces the unique stable matching that is simultaneously best possible for all residents [6] [7, Theorem 1.6.2]. These observations may be summarized as follows:

Theorem 1 *Given an instance of HR, the RGS algorithm constructs, in* $O(L)$ *time, the unique stable matching in which each assigned resident obtains the best hospital that he could obtain in any stable matching, while each unassigned resident is unassigned in every stable matching.*

A counterpart of the RGS algorithm, known as the *Hospital-oriented Gale/Shapley algorithm*, or HGS algorithm for short [7, Section 1.6.2], gives the unique stable matching that similarly satisfies an optimality property for the hospitals [7, Theorem 1.6.1].

Although there may be many stable matchings for a given instance I of HR, some key structural properties hold regarding unassigned residents and under-subscribed hospitals with respect to all stable matchings in I, as follows.

Theorem 2 *For a given instance of HR:*

- *The same residents are assigned in all stable matchings;*
- *Each hospital is assigned the same number of residents in all stable matchings;*
- *Any hospital that is under-subscribed in one stable matching is assigned exactly the same set of residents in all stable matchings.*

These results are collectively known as the "Rural Hospitals Theorem" (see [7, Section 1.6.4] for further details). Furthermore, the set of stable matchings in I forms a distributive lattice under a natural dominance relation [7, Section 1.6.5].

```
$M := \emptyset$;
while (some resident $r_i$ is unassigned and $r_i$ has a nonempty list) {
    $h_j$ := first hospital on $r_i$'s list;
    /* $r_i$ applies to $h_j$ */
    $M := M \cup \{(r_i, h_j)\}$;
    if ($h_j$ is over-subscribed) {
        $r_k$ := worst resident in $M(h_j)$ according to $h_j$'s list;
        $M := M \backslash \{(r_k, h_j)\}$;
    }
    if ($h_j$ is full) {
        $r_k$ := worst resident in $M(h_j)$ according to $h_j$'s list;
        for (each successor $r_l$ of $r_k$ on $h_j$'s list)
            delete the pair $(r_l, h_j)$;
    }
}
```

Hospitals/Residents Problem, Fig. 1 Gale/Shapley algorithm for HR

Applications

Practical applications of HR are widespread, most notably arising in the context of centralized automated matching schemes that assign applicants to posts (e.g., medical students to hospitals, school leavers to universities, and primary school pupils to secondary schools). Perhaps the largest and best-known example of such a scheme is the National Resident Matching Program (NRMP) in the USA [8], which annually assigns around 31,000 graduating medical students (known as residents) to their first hospital posts, taking into account the preferences of residents over hospitals and vice versa and the hospital capacities. Counterparts of the NRMP are in existence in other countries, including Canada [9] and Japan [10]. These matching schemes essentially employ extensions of the RGS algorithm for HR.

Centralized matching schemes based largely on HR also occur in other practical contexts, such as school placement in New York [1], university faculty recruitment in France [3], and university admission in Spain [16]. Further applications are described in [15, Section 1.3.7].

Indeed, the Nobel Prize in Economic Sciences was awarded in 2012 to Alvin Roth and Lloyd Shapley, partly for their theoretical work on HR and its variants [6, 18] and partly for their contribution to the widespread deployment of algorithms for HR in practical settings such as junior doctor allocation as noted above.

Extensions of HR

One key extension of HR that has considerable practical importance arises when an instance may involve a set of *couples*, each of which submits a joint preference list over pairs of hospitals (typically in order that the members of the couple can be located geographically close to one another). The extension of HR in which couples may be involved is denoted by HRC; the stability definition in HRC is a natural extension of that in HR (see [15, Section 5.3] for a formal definition of HRC). It is known that an instance of HRC need not admit a stable matching (see [4]). Moreover, the problem of deciding whether an HRC instance admits a stable matching is NP-complete [17].

HR may be regarded as a many-one generalization of SM. A further generalization of SM is to a many-many stable matching problem, in which both residents and hospitals may be multiply assigned subject to capacity constraints. In this case, residents and hospitals are more commonly referred to as *workers* and *firms*, respectively. There are two basic variations of the many-many stable matching problem according to whether workers rank (i) individual acceptable firms in order of preference and vice versa or (ii) acceptable *subsets* of firms in order of preference and vice versa. Previous work relating to both models is surveyed in [15, Section 5.4].

Other variants of HR may be obtained if preference lists include ties. This extension is again important from a practical perspective, since it may be unrealistic to expect a popular hospital to rank a large number of applicants in strict order, particularly if it is indifferent among groups of applicants. The extension of HR in which preference lists may include ties is denoted by HRT. In this context three natural stability definitions arise, the so-called *weak stability*, *strong stability*, and *super-stability* (see [15, Section 1.3.5] for formal definitions of these concepts). Given an instance I of HRT, it is known that weakly stable matchings may have different sizes, and the problem of finding a maximum cardinality weakly stable matching is NP-hard (see entry ▸ Stable Marriage with Ties and Incomplete Lists for further details). On the other hand, in contrast to the case for weak stability, a super-stable matching in I need not exist, though there is an $O(L)$ algorithm to find such a matching if one does [11]. Analogous results hold in the case of strong stability – in this case, an $O(L^2)$ algorithm [13] was improved by an $O(CL)$ algorithm [14] and extended to the many-many case [5]. Furthermore, counterparts of the Rural Hospitals Theorem hold for HRT under each of the super-stability and strong stability criteria [11, 19].

A further generalization of HR arises when each hospital may be split into several departments, where each department has a capacity, and residents rank individual departments in order of preference. This variant is modeled by the *Student-Project Allocation problem* [15, Section 5.5]. Finally, the *Hospitals/Residents problem under Social Stability* [2] is an extension of HR in which an instance is augmented by a *social network graph* G (a bipartite graph whose vertices correspond to residents and hospitals and whose edges form a subset of A) such that a blocking pair must additionally satisfy the property that it forms an edge of G. Edges in G correspond to resident–hospital pairs that are acquainted with one another and therefore more likely to block a matching in practice.

Open Problems

As noted in Section "Applications," ties in the hospitals' preference lists may arise naturally in practical applications. In an HRT instance, weak stability is the most commonly-studied stability criterion, due to the guaranteed existence of such a matching. Attempting to match as many residents as possible motivates the search for large weakly stable matchings. Several approximation algorithms for finding a maximum cardinality weakly stable matching have been formulated (see ▸ Stable Marriage with Ties and Incomplete Lists and [15, Section 3.2.6] for further details). It remains open to find tighter upper and lower bounds for the approximability of this problem.

URL to Code

Ada implementations of the RGS and HGS algorithms for HR may be found via the following URL: http://www.dcs.gla.ac.uk/research/algorithms/stable.

Cross-References

- ▸ Optimal Stable Marriage
- ▸ Ranked Matching
- ▸ Stable Marriage
- ▸ Stable Marriage and Discrete Convex Analysis
- ▸ Stable Marriage with Ties and Incomplete Lists
- ▸ Stable Partition Problem

Recommended Reading

1. Abdulkadiroğlu A, Pathak PA, Roth AE (2005) The New York city high school match. Am Econ Rev 95(2):364–367
2. Askalidis G, Immorlica N, Kwanashie A, Manlove DF, Pountourakis E (2013) Socially stable matchings in the Hospitals/Residents problem. In: Proceedings

of the 13th Algorithms and Data Structures Symposium (WADS'13), London, Canada. Lecture Notes in Computer Science, vol 8037. Springer, pp 85–96
3. Baïou M, Balinski M (2004) Student admissions and faculty recruitment. Theor Comput Sci 322(2):245–265
4. Biró P, Klijn F (2013) Matching with couples: a multidisciplinary survey. Int Game Theory Rev 15(2):Article number 1340008
5. Chen N, Ghosh A (2010) Strongly stable assignment. In: Proceedings of the 18th annual European Symposium on Algorithms (ESA'10), Liverpool, UK. Lecture Notes in Computer Science, vol 6347. Springer, pp 147–158
6. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
7. Gusfield D, Irving RW (1989) The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge, USA
8. http://www.nrmp.org (National Resident Matching Program website)
9. http://www.carms.ca (Canadian Resident Matching Service website)
10. http://www.jrmp.jp (Japan Resident Matching Program website)
11. Irving RW, Manlove DF, Scott S (2000) The Hospitals/Residents problem with Ties. In: Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT'00), Bergen, Norway. Lecture Notes in Computer Science, vol 1851. Springer, pp 259–271
12. Irving RW, Manlove DF, Scott S (2002) Strong stability in the Hospitals/Residents problem. Technical report TR-2002-123, Department of Computing Science, University of Glasgow. Revised May 2005
13. Irving RW, Manlove DF, Sott S (2003) Strong stability in the Hospitals/Residents problem. In: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03), Berlin, Germany. Lecture Notes in Computer Science, vol 2607. Springer, pp 439–450. Full version available as [12]
14. Kavitha T, Mehlhorn K, Michail D, Paluch KE (2007) Strongly stable matchings in time $O(nm)$ and extension to the Hospitals-Residents problem. ACM Trans Algorithms 3(2):Article number 15
15. Manlove DF (2013) Algorithmics of matching under preferences. World Scientific, Hackensack, Singapore
16. Romero-Medina A (1998) Implementation of stable solutions in a restricted matching market. Rev Econ Des 3(2):137–147
17. Ronn E (1990) NP-complete stable matching problems. J Algorithms 11:285–304
18. Roth AE, Sotomayor MAO (1990) Two-sided matching: a study in game-theoretic modeling and analysis. Econometric society monographs, vol 18. Cambridge University Press, Cambridge/New York, USA
19. Scott S (2005) A study of stable marriage problems with ties. PhD thesis, Department of Computing Science, University of Glasgow

Hospitals/Residents Problems with Quota Lower Bounds

Huang Chien-Chung
Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

Keywords

Hospitals/Residents problem; Lower quotas; Stable matching

Years and Authors of Summarized Original Work

2010; Biró, Fleiner, Irving, Manlove;
2010; Huang;
2011; Hamada, Iwama, Miyazaki;
2012; Fleiner, Kamiyama

Problem Definition

The Hospitals/Residents (**HR**) problem is the many-to-one version of the stable marriage problem introduced by Gale and Shapley. In this problem, a bipartite graph $G = (\mathcal{R} \cup \mathcal{H}, E)$ is given. Each vertex in $\mathcal{H}$ represents a hospital and each vertex in $\mathcal{R}$ a resident. Each vertex has a preference over its neighboring vertices. Each hospital h has an upper quota $u(h)$ specifying the maximum number of residents it can take in a matching. The goal is to find a stable matching while respecting the upper quotas of the hospitals.

The original **HR** has been well studied in the past decades. A recent trend is to assume that each hospital h also comes with a lower quota $l(h)$. In this context, it is required (if possible) that a matching satisfies both the upper and the lower quotas of each hospital. The introduction of such lower quotas is to enforce some policy in hiring or to make the outcome more fair. It is well-known that hospitals in some rural areas suffer from the shortage of doctors.

With the lower quotas, the definition of stability in **HR** and the objective of the problem depend on the applications. Below we summarize three variants that have been considered in the literature.

Minimizing the Number of Blocking Pairs

In this variant, a matching M is feasible if, for each hospital h, $l(h) \leq |M(h)| \leq u(h)$. Given a feasible matching, a resident r and a hospital h form a blocking pair if the following condition holds. (i) $(r, h) \in E \backslash M$, (ii) r is unassigned in M or r prefers h to his assignment $M(r)$, and (iii) $|M(h)| < u(h)$ or h prefers r to one of its assigned residents. A matching is stable if the number of blocking pairs is 0. It is straightforward to check whether a stable matching exists. We assume that the given instance has no stable matching and the objective is to find a matching with the minimum number of blocking pairs. We call this problem **Min-BP HR**. An alternative objective is to minimize the number of residents that are part of a blocking pair in a matching. We call this problem **Min-BR HR**.

HR with the Option of Closing a Hospital

The following variation of **HR** is motivated by the higher education system in Hungary. Instead of requiring all hospitals to have enough residents to meet their lower quotas, it is allowed that a hospital be closed as long as there is not too much demand for it.

Precisely, in this variant, a matching M is feasible if, for each hospital h, $|M(h)| = 0$ or $l(h) \leq |M(h)| \leq u(h)$. In the former case, a hospital is closed; in the latter case, a hospital is opened. Given a feasible matching M, it is stable if

1. There is no opened hospital h and resident r so that (i) $(h, r) \in E \backslash M$, (ii) r is unassigned in M or r prefers h to his assignment $M(r)$, and (iii) $|M(h)| < u(h)$ or h prefers r to one of its assigned residents;
2. There is no closed hospital h and a set $R \subseteq \mathcal{R}$ of residents so that (i) $|R| \geq |l(h)|$, (ii) for each $r \in R$, $(r, h) \in E \backslash M$, and (iii) each resident $r \in R$ is either unassigned or prefers h to his assigned hospital $M(r)$.

With the above definition of stability, we refer to the question of the existence of a stable matching as **HR woCH**.

Classified HR

Motivated by the practice in academic hiring, Huang introduced a more generalized variant of **HR**. In this variant, a hospital h has a classification $h_{\mathcal{C}}$ over its neighboring residents. Each class $C \in h_{\mathcal{C}}$ comes with a upper quota $u(C)$ and a lower quota $l(C)$. A matching M is feasible if, for each hospital h and for each of its classes $c \in h_{\mathcal{C}}$, $l(C) \leq |M(h)| \leq u(C)$. A feasible matching M is stable if the following condition holds: there is no hospital h such that

1. There exists a resident r so that $(r, h) \in E \backslash M$, and r is either unassigned in M or r prefers h to his assignment $M(r)$;
2. For every class $C \in h_{\mathcal{C}}$, $l(C) \leq |M(h) \cup \{r\}| \leq u(C)$, or there exists another resident $r' \in M(h)$ so that h prefers r to r' and for every class $C \in h_{\mathcal{C}}$, $l(C) \leq |M(h) \cup \{r\} \backslash \{r'\}| \leq u(C)$.

With the above definition of stability, we refer to the question of the existence of a stable matching as **CHR**.

Key Results

For the first variant where the objective is to minimize the number of blocking pairs, Hamada et al. showed the following tight results.

Theorem 1 ([3]) *For any positive constant $\epsilon > 0$, there is no polynomial-time $(|\mathcal{R}| + |\mathcal{H}|)^{1-\epsilon}$-approximation algorithm for **Min-BP HR** unless P=NP. This holds true even if the given bipartite graph is complete and all upper quotas are 1 and all lower quotas are 0 or 1.*

Theorem 2 ([3]) *There is a polynomial-time* $(|\mathcal{R}| + |\mathcal{H}|)$*-approximation algorithm for* ***Min-BP HR****.*

In the case that the objective is to minimize the number of residents involved in blocking pairs, Hamada et al. showed the following.

Theorem 3 ([3]) ***Min-BR HR*** *is NP-hard. This holds true even if the given bipartite graph is complete and all hospitals have the same preference over the residents.*

Theorem 4 ([3]) *There is a polynomial-time* $\sqrt{|\mathcal{R}|}$*-approximation algorithm for* ***Min-BR HR****.*

For the second variant, where a hospital is allowed to be closed, Biró et al. showed the following.

Theorem 5 ([1]) *The problem* ***HR woCH*** *is NP-complete. This holds true even if all upper quotas are at most 3.*

For the last variant where each hospital is allowed to classify the neighboring residents and sets the upper and lower quotas for each of its classes, Huang showed that if all classifications of the hospitals are laminar families, the problem is in P. Fleiner and Kamiyama later proved the same result by a significantly simpler matroid-based technique.

Theorem 6 ([2,4]) *In* ***CHR****, if all classifications of the hospitals are laminar families, then one find a stable matching or detect its absence in the given instance in* $O(nm)$ *time, where* $n = |\mathcal{R} \cup \mathcal{H}|$ *and* $m = |E|$*.*

Recommended Reading

1. Biro P, Fleiner T, Irving RW, Manlove DF (2010) The College Admissions problem with lower and common quotas. Theor Comput Sci 411(34–36):3136–3153
2. Fleiner T, Kamiyama N (2012) A matroid approach to stable matchings with lower quotas In: 23nd annual ACM-SIAM symposium on discrete algorithms, Kyoto, pp 135–142
3. Hamada K, Iwama K, Miyazaki S (2011) The Hospitals/Residents problem with quota lower bounds. In: 19th annual European symposium on algorithms, Saarbrücken, vol 411(34–36), pp 180–191
4. Huang C-C (2010) Classified stable matching. In: 21st annual ACM-SIAM symposium on discrete algorithms, Austin, pp 1235–1253

Hub Labeling (2-Hop Labeling)

Daniel Delling[1], Andrew V. Goldberg[2], and Renato F. Werneck[3]
[1]Microsoft, Silicon Valley, CA, USA
[2]Microsoft Research – Silicon Valley, Mountain View, CA, USA
[3]Microsoft Research Silicon Valley, La Avenida, CA, USA

Keywords

Distance oracles; Labeling algorithms; Shortest paths

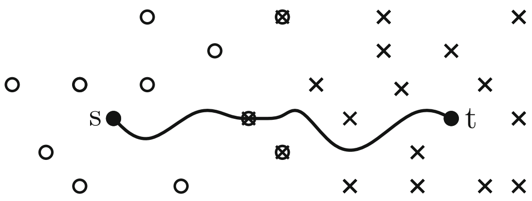
Years and Authors of Summarized Original Work

2003; Cohen, Halperin, Kaplan, Zwick
2012; Abraham, Delling, Goldberg, Werneck
2013; Akiba, Iwata, Yoshida
2014; Delling, Goldberg, Pajor, Werneck
2014; Delling, Goldberg, Savchenko, Werneck

Problem Definition

Given a directed graph $G = (V, A)$ (with $n = |V|$ and $m = |A|$) with a length function $\ell : A \to \mathbf{R}^+$ and a pair of vertices s, t, a *distance oracle* returns the distance $\text{dist}(s, t)$ from s to t. A *labeling algorithm* [18] implements distance oracles in two stages. The *preprocessing* stage computes a *label* for each vertex of the input graph. Then, given s and t, the *query* stage computes $\text{dist}(s, t)$ using only the labels of s and t; the query does not explicitly use G and ℓ.

Hub labeling (HL) (or 2-hop labeling) is a special kind of labeling algorithm. The label $L(v)$ of a vertex v consists of two parts: the *forward label* $L_f(v)$ is a collection of vertices w with their distances $\text{dist}(v, w)$ from v, while the



Hub Labeling (2-Hop Labeling), Fig. 1 Example of a hub labeling. The hubs of s are *circles*; the hubs of t are *crosses* (Taken from [3])

backward label $L_b(v)$ is a collection of vertices u with their distances $\text{dist}(u, v)$ to v. (If the graph is undirected, a single label per vertex suffices.) The vertices in v's label are the *hubs* of v. The labels must obey the *cover property*: for any two vertices s and t, the set $L_f(s) \cap L_b(t)$ must contain at least one hub that is on the shortest $s - t$ path. Given the labels, HL queries are straightforward: to find $\text{dist}(s, t)$, simply find the hub $x \in L_f(s) \cap L_b(t)$ that minimizes $\text{dist}(s, x) + \text{dist}(x, t)$ (see Fig. 1 for an example). If the hubs in each label are sorted by ID, queries consist of a simple linear sweep over the labels, as in mergesort.

The size of a forward (backward) label, $|L_f(v)|$ ($|L_b(v)|$), is the number of hubs it contains. The *size of a labeling* L is the sum of the average label sizes, $(L_f(v) + L_b(v))/2$, over all vertices. The memory footprint of the algorithm is proportional to the size of the labeling, while query times are determined by the maximum label size. Queries themselves are trivial; the hard part is an efficient implementation of a preprocessing algorithm that, given G and ℓ, computes a small hub labeling.

Key Results

We describe an approximation algorithm for finding labelings of size within $O(\log n)$ of the optimal [9], as well as its generalization to other objectives, including the maximum label size [6]. Although polynomial, these approximation algorithms do not scale to large networks. For more practical alternatives, we discuss *hierarchical hub labelings* (HHLs), a subclass of HL. We show that HHLs are closely related to vertex orderings and present efficient algorithms for computing the minimal HHL for a given ordering, as well as heuristics for finding vertex orderings that lead to small labels. In particular, the RXL algorithm uses sampling to efficiently approximate a greedy vertex order, leading to empirically small labels. RXL can handle large problems from several application domains. We then discuss representations of hub labels that allow various trade-offs between space and query time.

General Hub Labelings

The time and space efficiency of the distance oracles we discuss depend on the label size. If labels are big, HL is impractical. Gavoille et al. [15] show that there exist graphs for which general labelings must have size $\tilde{\Theta}(n^2)$. For planar graphs, they give an $\tilde{\Omega}(n^{4/3})$ lower and $\tilde{O}(n^{3/2})$ upper bound. They also show that graphs with k-separators have hub labelings of size $\tilde{O}(nk)$. Abraham et al. [1] show that graphs with small highway dimension (which they conjecture include road networks) have small hub labelings.

Given a particular graph, computing a labeling with the smallest size is NP-hard. Cohen et al. [9] developed an $O(\log n)$-approximation algorithm for the problem. Next we discuss this *general HL* (GHL) algorithm.

A *partial labeling* is a labeling that does not necessarily satisfy the cover property. Given a partial labeling $L = (L_f, L_b)$, we say that a vertex pair $[u, w]$ is *covered* if $L_f(u) \cap L_b(w)$ contains a vertex on a shortest path from u to w and *uncovered* otherwise. GHL maintains a partial labeling L (initially empty) and the corresponding set U of uncovered vertex pairs. Each iteration of the algorithm selects a vertex v and two subsets $X', Y' \subseteq V$, adds $(v, \text{dist}(x, v))$ to $L_f(x)$ for all $x \in X'$, and adds $(y, \text{dist}(v, y))$ to $L_b(y)$ for all $y \in Y'$. Then, GHL deletes from U the set $U(v, X', Y')$ of vertex pairs that become covered by this augmentation. Among all $v \in V$ and $X', Y' \subseteq V$, the triple (v, X', Y') picked in each iteration is one that maximizes $|U(v, X', Y')|/(|X'| + |Y'|)$, i.e., the ratio of the number of paths covered over the increase in label size.

Cohen et al.'s efficient implementation of GHL uses the notion of *center graphs*. Given a set U of vertex pairs and a vertex v, the center graph $G_v = (X, Y, A_v)$ is a bipartite graph with $X = Y = V$ such that an arc $(u, w) \in A_v$ if $[u, w] \in U$ and some shortest path from u to w in G go through v. If U is the set of uncovered vertex pairs, then, for a fixed vertex v, maximizing $|U(v, X', Y')|/(|X'| + |Y'|)$ over all $X', Y' \subseteq V$ is (by definition) the same as finding the vertex induced-subgraph of G_v with maximum *density* (defined as its number of arcs divided by its number of vertices). This *maximum density subgraph* (MDS) problem can be solved in polynomial time using parametric flows (see e.g., [14]). To maximize the ratio over all triples (v, X', Y'), GHL solves an MDS problem for center graphs G_v and picks the densest of the n resulting subgraphs. It then adds the corresponding vertex v^* to the labels of the vertices given by the sides of the MDS. Arcs corresponding to newly covered pairs are removed from center graphs between iterations.

Cohen et al. show that GHL is a special case of the greedy set cover algorithm [8] and thus gives an $O(\log n)$-optimal labeling. They also show that the same guarantee holds if one uses a constant-factor approximation to the MDS. We refer to a k approximation of MDS as a k-AMDS. Using a linear-time 2-AMDS algorithm by Kortsarz and Peleg [17], each GHL iteration is dominated by n AMDS computations on graphs with $O(n^2)$ arcs. Since each iteration increases the size of the labeling, the number of iterations is at most $O(n^2)$. The total running time of GHL is thus $O(n^5)$.

Delling et al. [11] improve the time bound for GHL to $O(n^3 \log n)$ using *eager* and *lazy* evaluation. Intuitively, eager evaluation finds an AMDS G' of G such that deleting G' reduces the MDS value of G by a constant factor. More precisely, given a graph G, an upper bound μ on the MDS value of G and a parameter $\alpha > 1$, *α-eager evaluation* attempts to find a (2α)-AMDS G' of G such that the MDS value of G with the arcs of G' deleted is at most μ/α. If the evaluation fails to find such G', the MDS value of G is at most μ/α. *Lazy evaluation* was introduced by Cohen et al. [9] to speed up their implementation of GHL and refined by Stengel et al. [20]. It is based on the observation that the MDS value of a center graph does not increase as the algorithm adds vertices to labels and removes arcs from center graphs.

The eager-lazy algorithm maintains upper bounds on the center subgraph densities μ_v computed in previous iterations. These values are computed during initialization and updated in a lazy fashion as follows. In each iteration, the algorithm picks the maximum μ_v and applies α-eager evaluation to G_v. If the evaluation succeeds, the labels are updated. Regardless of whether the evaluation succeeds or not, μ_v/α is a valid upper bound on the density of G_v at the end of the iteration. This can be used to show that each vertex is selected by $O(n \log n)$ iterations, each taking $O(n^2)$ time.

Babenko et al. [6] generalize the definition of a labeling size as follows. Suppose vertex IDs are $1, 2, \ldots, n$. Define a $(2n)$-dimensional vector $\mathcal{L}$ by $\mathcal{L}_{2i-1} = |L_f(i)|$ and $\mathcal{L}_{2i} = |L_b(i)|$. The p-norm of $\mathcal{L}$ is defined as $\|\mathcal{L}\|_p = (\sum_{i=0}^{2n-1} \mathcal{L}_i^p)^{1/p}$, where p is a natural number and $\|\mathcal{L}\|_\infty = \max \mathcal{L}_i$. Note that $\|\mathcal{L}\|_1/2$ is the total size of the labeling and $\|\mathcal{L}\|_\infty$ is the maximum label size. Babenko et al. [6] generalize the algorithm of Cohen et al. to obtain an $O(\log n)$-approximation algorithm for this more general problem in $O(n^5)$ time. Delling et al. [11] show that the eager-lazy approach yields an $O(\log n)$-approximation algorithm running in time $O(n^3 \log n \min(p, \log n))$.

Hierarchical Hub Labelings

Even with the performance improvements mentioned above, GHL requires too much time and space to work on large networks. To overcome this problem, one may use heuristics that have no known theoretical guarantees on the label size but produce small labels for large instances from a wide variety of domains. The most successful current heuristics use a restricted class of labelings called *hierarchical hub labeling* (HHL) [4]. Hierarchical labels have the cover property and implement exact distance oracles.

Given a labeling, let $v \lesssim w$ if w is a hub of $L(v)$. HL is *hierarchical* if $\lesssim$ is a partial order. (Intuitively, $v \lesssim w$ if w is "more important" than v.) We say that an HHL *respects* a given (total) order on the vertices if the partial order $\lesssim$ induced by the HHL is consistent with the order.

Consider an order defined by a permutation *rank*, with $rank(v) < rank(w)$ if v appears before (is less important than) w. The *canonical labeling* L for *rank* is defined as follows [4]. Vertex v belongs to $L_f(u)$ if and only if there exists w such that v is the highest-ranked vertex that hits $[u, w]$. Similarly, v belongs to $L_b(w)$ if and only if there exists u such that v is the highest-ranked vertex that hits $[u, w]$.

Abraham et al. [4] prove that the canonical labeling for a given vertex order *rank* is the minimum-sized labeling that respects *rank*. This suggests a two-stage approach for finding a small hierarchical hub labeling: first, find a "good" vertex order, and then compute its corresponding canonical labeling. We first discuss the latter step and then the former.

From Orderings to Labelings

We first consider how, given an order *rank*, one can compute the canonical hierarchical labeling L that respects *rank*.

The straightforward way is to just apply the definition: for every pair $[u, w]$ of vertices, find the maximum-ranked vertex on any shortest u–w path, and then add it to $L_f(u)$ and $L_b(w)$. Although polynomial, this algorithm is too slow in practice.

A faster (but still natural) algorithm is as follows [4]. Start with an empty (partial) labeling L, and process vertices from the most to least important. When processing v, for every uncovered pair $[u, w]$ that v covers, add v to $L_f(u)$ and $L_b(w)$. (In other words, add v to the labels of all end points of arcs in the center graph G_v.) Abraham et al. [4] show how to implement this in $O(mn \log n)$ time and $\Theta(n^2)$ space, which is still impractical for large instances.

When labels are not too large, a much more efficient solution is the *pruned labeling* (PL) algorithm by Akiba et al. [5]. Starting from empty labels, PL also processes vertices from the most to least important, with the iteration that processes vertex v, adding v to all relevant labels. The crucial observation is that, when processing v, one only needs to look at uncovered pairs containing v itself; if $[u, v]$ is not covered, PL adds v to $L_f(u)$; if $[v, w]$ is not covered, it adds v to $L_b(w)$. This is enough because of the subpath optimality property of the shortest paths.

To process v efficiently, PL runs two pruned Dijkstra searches [13] from v. The first search works on the forward graph (out of v) as follows. Before scanning a vertex w (with distance label $d(w)$ within the Dijkstra search), it computes a v–w distance estimate q by performing an HL query with the current partial labels. (If the labels do not intersect, set $q = \infty$.) If $q \le d(w)$, the $[v, w]$ pair is already covered by previous hubs, so PL prunes the search (ignores w). Otherwise (if $q > d(w)$), PL adds $(v, \mathrm{dist}(v, w))$ to $L_b(w)$ and scans w as usual. The second Dijkstra search uses the reverse graph and is pruned similarly; it adds $(v, \mathrm{dist}(w, v))$ to $L_f(w)$ for all scanned vertices w. Note that the number of Dijkstra scans equals the size of the labeling. Since each visited vertex requires an HL query using partial labels, the running time can be quadratic in the average label size. It is easy to see that PL produces canonical labelings.

The final algorithm we discuss, due to Abraham et al. [4], computes a hierarchical hub labeling from a vertex ordering recursively. Its basic building block is the *shortcut operation* (see e.g., [16]). To shortcut a vertex v, the operation deletes v from the graph and adds arcs to ensure that the distances between the remaining vertices remain unchanged. For every pair consisting of an incoming arc (u, v) and an outgoing arc (v, w), the algorithm checks if $(u, v) \cdot (v, w)$ is the only shortest u–w path (by running a partial Dijkstra search from u or w) and, if so, adds a new arc (u, w) with length $\ell(u, w) = \ell(u, v) + \ell(v, w)$.

The recursive algorithm computes one label at a time, from the bottom up (from the least to the most important vertex). It starts by shortcutting the least important vertex v from G to get a graph

G' (same as G, but without v and its incident arcs and with the added shortcuts). It then recursively finds a labeling for G', which gives correct distances (in G) for all pairs of vertices not containing v. Then, the algorithm computes the label of v from the labels of its neighbors. We describe how to compute $L_f(v)$; $L_b(v)$ is computed similarly. The crucial observation is that any nontrivial shortest path starting at v must go through one of its neighbors. Accordingly, we initialize $L_f(v)$ with entry $(v, 0)$ (to cover the trivial path from v to itself), and then, for every neighbor w of v in G and every entry $(x, \text{dist}(w, x)) \in L_f(w)$, add $(x, \ell(v, w) + \text{dist}(w, x))$ to $L_f(v)$. If x already is a hub of v, we only keep the smallest entry for x. Finally, we prune from $L_f(v)$ the entries $(x, \ell(v, w) + \text{dist}(w, x))$ for which $\ell(v, w) + \text{dist}(w, x) > \text{dist}(v, x)$. (This can happen if the shortest path from v to x through another neighbor w' of v is shorter than the one through w.) Note that $\text{dist}(v, x)$ can be computed using the labels of v and x. In general, the shortcut operation can make the graph dense, limiting the efficiency of the bottom-up approach. On some network classes, such as road networks, the graph remains sparse and the approach scales to large problems.

Vertex Ordering Heuristics

As mentioned above, the size of the labeling is determined by the ordering. The most natural approach to capture the notion of importance is attributed to Abraham et al. [4], whose *greedy ordering algorithm* obtains good orderings on a wide class of problems. It orders vertices from the most to least important using a greedy selection rule. In each iteration, it selects as the next most important hub the vertex v that hits the most vertex pairs not covered by previously selected vertices.

When the shortest paths are unique, this can be implemented relatively efficiently. The algorithm maintains (initially full) the shortest-path trees from each vertex in the graph. The tree T_s rooted at s implicitly represents all shortest paths starting at s. The total number of descendants of a vertex v (in aggregate over all trees) is exactly the number of paths it covers. Once such a vertex v is picked as the next hub, we restore this invariant for the remaining paths by removing all of v's descendants (including v itself) from all trees. Abraham et al. [4] show how the entire greedy order can be found in $O(nm \log n)$ time. An alternative algorithm (in the same spirit) works even if the shortest paths are not unique, but takes $O(n^3)$ time [12].

The *weighted greedy ordering algorithm* is similar but selects v so as to maximize the ratio of the number of uncovered paths that v covers to the increase in the label size if v is selected next. This gives slightly better results and can be implemented in the same time bounds as the greedy ordering algorithm [4, 12]. Although faster than GHL, none of these greedy variants scale to large graphs.

To cope with this problem, Delling et al. [12] developed RXL (Robust eXact Labeling), which can be seen as a sampling version of the greedy ordering algorithm. In each iteration, RXL finds a vertex v that *approximately* maximizes the number of pairs covered. Rather than maintaining n shortest-path trees, RXL maintains shortest-path trees from a small number of roots picked uniformly at random. It estimates the coverage of v based on how many descendants it has in these trees. To reduce the bias in this estimation, the algorithm discards outliers before taking the average number of descendants. Moreover, as the original trees shrink (because some of its subtrees become covered), new subtrees (from other roots) are added. These new trees are not full, however; they are pruned from the start (using PL), ensuring the total space (and time) usage remains under control.

For certain graph classes, simpler ordering techniques can be used. Akiba et al. [5] show that ordering by degree works well on a subclass of complex networks. Abraham et al. [2,4] show that the order induced by the contraction hierarchies (CH) algorithm [16] works well on road networks and other sparse inputs. CH order vertices from the bottom up: using only local information, it determines the least important vertex, shortcuts it, and repeats the process in the remaining graph. The most relevant signals to estimate the importance of v are the arc difference (of number of arcs removed and added if v were shortcut) and how many neighbors of v have already been shortcut.

Label Representation and Queries

Given a source s and a target t, one can compute the minimum of $\text{dist}(s, v) + \text{dist}(v, t)$ over all $v \in L_f(s) \cap L_b(t)$ in $O(|L_f(s)| + |L_f(t)|)$ time. If vertex labels are represented as arrays sorted by hub IDs, one can compute $L_f(s) \cap L_b(t)$ by a coordinated sweep of the corresponding arrays, as in mergesort. This is very cache efficient and works well when the two labels have similar sizes.

In some applications, label sizes can be very different. Assuming (without loss of generality) that $|L_f(s)| \ll |L_f(t)|$, one can compute $L_f(s) \cap L_b(t)$ in time $O(|L_f(s)| + \log(|L_b(t)|))$ by performing a binary search for each hub $v \in L_f(s)$ to determine if v is in $L_b(t)$. In fact, this *set intersection problem* can be solved even faster, in $O(\min(|L_f(s)|, |L_b(t)|))$ time [19].

As each label can be stored in a contiguous memory block, HL queries are well suited for an external memory (or even distributed) implementations, including relational databases [3] or key-value stores. In such cases, query times depend on the time to fetch two blocks of data.

For in-memory implementations of HL, storage may be a bottleneck. One can trade space for time using label compression, which interprets each label as a tree and stores common subtrees only once; this reduces space consumption by an order of magnitude, but queries become much less cache efficient [10, 12]. Another technique to reduce the space consumption is to store vertices and a constant number of their neighbors as superhubs in the labels [5]; on unweighted and undirected graphs, distances from a vertex v to all elements of a superhub can be represented compactly in difference form. This works well on some social and communication networks [5].

HL has efficient extensions to problems beyond point-to-point shortest paths, including one-to-many and via-point queries. These are important for applications in road networks, such as finding the closest points of interest, ride sharing, and path prediction [3].

Experimental Results

Even for very small (constant) sample sizes, the labels produced by RXL are typically no more than about 10 % bigger [12] than those produced by the full greedy hierarchical algorithms, which in turn are not much worse than those produced by GHL [11]. Scalability is much different, however. In a few hours in a modern CPU, GHL can only handle graphs with about 10,000 vertices [11]; for the greedy hierarchical algorithms, the practical limit is about 100,000 [4]. In contrast, as long as labels remain small, RXL scales to problems with millions of vertices [12] from a wide variety of graph classes, including meshes, grids, random geometric graphs (sensor networks), road networks, social networks, collaboration networks, and web graphs. For example, for a web graph with 18.5 million vertices and almost 300 million arcs, one can find labels with fewer than 300 hubs on average in about half a day [12]; queries then take less than 2 µs.

For some graph classes, other methods have faster preprocessing. For continental road networks with tens of millions of vertices, a hybrid approach combining weighted greedy (for the top few thousand vertices) with the CH order (for all other vertices) provides the best trade-off between preprocessing times and label size [2, 4]. On a benchmark data set representing Western Europe (about 18 million vertices, 42.5 million arcs), it takes roughly an hour to compute labels with about 70 hubs on average, leading to average query times of about 0.5 µs, roughly the time of ten random memory accesses. With additional improvements, one can further reduce query times (but not the label sizes) by half [2], making it the fastest algorithm for this application [7]. For some unweighted and undirected complex (social, communication, and collaboration) networks, simply sorting vertices by degree [5] produces labels that are not much bigger than those computed by a more sophisticated ordering technique.

Overall, RXL is the most robust method. For all instances tested in the literature, its preprocessing is never much slower than any other methods (and often much faster), and query times are similar. In particular, CH-based ordering is too costly for large complex networks (as contraction tends to create dense graphs), and the degree-based order leads to prohibitively large labels for road networks and web graphs.

Recommended Reading

1. Abraham I, Fiat A, Goldberg AV, Werneck RF (2010) Highway dimension, shortest paths, and provably efficient algorithms. In: Proceedings of 21st ACM-SIAM symposium on discrete algorithms, Austin, pp 782–793
2. Abraham I, Delling D, Goldberg AV, Werneck RF (2011) A hub-based labeling algorithm for shortest paths on road networks. In: Proceedings of the 10th international symposium on experimental algorithms (SEA'11), Chania. Volume 6630 of Lecture notes in computer science. Springer, pp 230–241
3. Abraham I, Delling D, Fiat A, Goldberg AV, Werneck RF (2012) HLDB: location-based services in databases. In: Proceedings of the 20th ACM SIGSPATIAL international symposium on advances in geographic information systems (GIS'12), Redondo Beach. ACM, pp 339–348
4. Abraham I, Delling D, Goldberg AV, Werneck RF (2012) Hierarchical hub labelings for shortest paths. In: Proceedings of the 20th annual European symposium on algorithms (ESA'12), Ljubljana. Volume 7501 of Lecture notes in computer science. Springer, pp 24–35
5. Akiba T, Iwata Y, Yoshida Y (2013) Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD'13, New York. ACM, pp 349–360
6. Babenko M, Goldberg AV, Gupta A, Nagarajan V (2013) Algorithms for hub label optimization. In: Fomin FV, Freivalds R, Kwiatkowska M, Peleg D (eds) Proceedings of 30th ICALP, Riga. Lecture notes in computer science, vol 7965. Springer, pp 69–80
7. Bast H, Delling D, Goldberg AV, Müller–Hannemann M, Pajor T, Sanders P, Wagner D, Werneck RF (2014) Route planning in transportation networks. Technical report MSR-TR-2014-4, Microsoft research
8. Chvátal V (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4(3): 233–235
9. Cohen E, Halperin E, Kaplan H, Zwick U (2003) Reachability and distance queries via 2-hop labels. SIAM J Comput 32:1338–1355
10. Delling D, Goldberg AV, Werneck RF (2013) Hub label compression. In: Proceedings of the 12th international symposium on experimental algorithms (SEA'13), Rome. Volume 7933 of Lecture notes in computer science. Springer, pp 18–29
11. Delling D, Goldberg AV, Savchenko R, Werneck RF (2014) Hub labels: theory and practice. In: Proceedings of the 13th international symposium on experimental algorithms (SEA'14), Copenhagen. Lecture notes in computer science. Springer
12. Delling D, Goldberg AV, Pajor T, Werneck RF (2014, to appear) Robust distance queries on massive networks. In: Proceedings of the 22nd annual European symposium on algorithms (ESA'14), Wroclaw. Lecture notes in computer science. Springer
13. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1: 269–271
14. Gallo G, Grigoriadis MD, Tarjan RE (1989) A fast parametric maximum flow algorithm and applications. SIAM J Comput 18:30–55
15. Gavoille C, Peleg D, Pérennes S, Raz R (2004) Distance labeling in graphs. J Algorithms 53(1): 85–112
16. Geisberger R, Sanders P, Schultes D, Vetter C (2012) Exact routing in large road networks using contraction hierarchies. Transp Sci 46(3):388–404
17. Kortsarz G, Peleg D (1994) Generating sparse 2-spanners. J Algorithms 17:222–236
18. Peleg D (2000) Proximity-preserving labeling schemes. J Graph Theory 33(3):167–176
19. Sanders P, Transier F (2007) Intersection in integer inverted indices. SIAM, Philadelphia, pp 71–83
20. Schenkel R, Theobald A, Weikum G (2004) HOPI: an efficient connection index for complex XML document collections. In: Advances in database technology – EDBT 2004. Springer, Berlin/Heidelberg, pp 237–255

Huffman Coding

Alistair Moffat
Department of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia

Keywords

Compression; Huffman code; Minimum-redundancy code

Years and Authors of Summarized Original Work

1952; Huffman
1976; van Leeuwen
1995; Moffat, Katajainen

Problem Definition

A sequence of n positive weights or frequencies is given, $\langle w_i > 0 \mid 0 \leq i < n\rangle$, together with an output radix r, with $r = 2$ in the case of binary output strings.

Objective To determine a sequence of integral codeword lengths $\langle \ell_i \mid 0 \leq i < n\rangle$ such that: (a) $\sum_{i=0}^{n-1} r^{-\ell_i} \leq 1$, and (b) $C = \sum_{i=0}^{n-1} \ell_i \cdot w_i$ is minimized. Any sequence of codeword lengths $\langle \ell_i \rangle$ that satisfies these two properties describes a *minimum-redundancy code* for the weights $\langle w_i \rangle$. Once a set of minimum-redundancy codeword lengths $\langle \ell_i \rangle$ has been identified, a prefix-free r-ary code in which symbol i is assigned a codeword of length ℓ_i can always be constructed.

Constraints

1. **Long messages.** In one application, each weight w_i is the frequency of symbol i in a message M of length $m = |M| = \sum_{i=0}^{n-1} w_i$, and C is the number of symbols required by a compressed representation of M. In this application it is usual to assume that $m \gg n$.
2. **Entropy-based limit.** Define $W = \sum_{i=0}^{n-1} w_i$ to be the sum of the weights and $p_i = w_i / W$ to be the corresponding probability of symbol i. Define $H_0 = -\sum_{i=0}^{n-1}(p_i \log_2 p_i)$ to be the zero-order entropy of the distribution. Then when $r = 2$, $\lceil nH_0 \rceil \leq C \leq n\lceil \log_2 n \rceil$.

Key Results

A minimum-redundancy code can be identified in $O(n)$ time if the weights w_i are nondecreasing and in $O(n \log n)$ time if the weights must be sorted first.

Example Weights

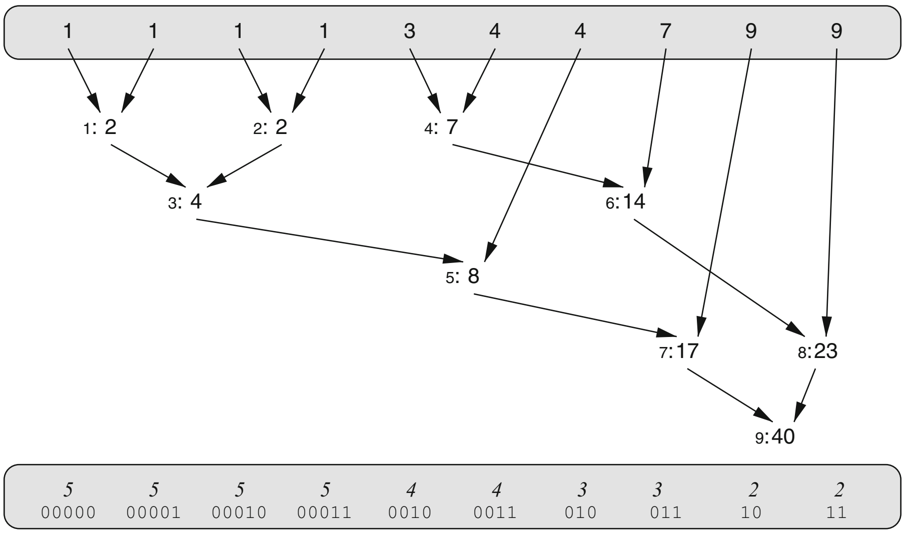
The $n = 10$ weights $\langle 1, 1, 1, 1, 3, 4, 4, 7, 9, 9\rangle$ with $W = 40$ are used as an example.

Huffman's Algorithm

In 1952 David Huffman [3] described a process for calculating minimum-redundancy codes, developed in response to a term-paper challenge set the year before by his MIT class instructor, Robert Fano, a problem that Fano and his collaborator Claude Shannon had already tackled unsuccessfully [7]. In his solution Huffman created a classic algorithm that is taught to most undergraduate computing students as part of algorithms classes. Initially the sequence of input weights $\langle w_i \rangle$ is regarded as being the leaves of a tree, with no internal nodes, and each leaf the root of its own subtree. The two subtrees (whether singleton leaves or internal nodes) with the smallest root nodes are then combined by making both of them children of a new parent, with an assigned weight calculated as the sum of the two original nodes. The pool of subtrees decreases by one at each cycle of this process; after $n-1$ iterations a total of $n-1$ internal nodes has been added, and all of the original nodes must be leaves in a single tree and descendants of that tree's root node.

Figure 1 shows an example of codeword length computation, with the original weights across the top. Each iteration takes the two least-weight elements (leaf or internal) and combines them to make a new internal node; note that the internal nodes are created in nondecreasing weight order. Once the Huffman tree has been constructed, the sequence $\langle \ell_i \rangle$ can be read from it, by computing the depth of each corresponding leaf node. In Fig. 1, for example, one of the elements of weight 4 is at depth three in the tree, and one is at depth four from the root, hence $\ell_5 = 4$ and $\ell_6 = 3$. A set of codewords can be assigned at the same time as the depths are being computed; one possible assignment of codewords that satisfies the computed sequence $\langle \ell_i \rangle$ is shown in the second row in the lower box. Decoding throughput is considerably faster if codewords are assigned systematically based on codeword length in the manner shown, rather than by strictly following the edge labeling of the Huffman tree from which the codeword lengths were extracted [6].

Because ties can occur and can be broken arbitrarily, different codes are also possible. The sequence $\langle \ell_i \rangle = \langle 6, 6, 6, 6, 4, 3, 3, 3, 2, 2\rangle$ has the same cost of $C = 117$ bits as the one shown



Huffman Coding, Fig. 1 Example of (binary) codeword lengths calculated using Huffman's algorithm, showing the order in which internal nodes are formed, and their weights. The input weights in the top section are used to compute the corresponding codeword lengths in the bottom box. A valid assignment of prefix-free codewords is also shown

in the figure. For the example weights, $H_0 = 2.8853$ bits per symbol, providing a lower bound of $\lceil 115.41 \rceil = 116$ bits on the total cost C for the input weights. In this case, the minimum-redundancy codes listed are just 1 bit inferior to the entropy-based lower limit.

Implementing Huffman's Algorithm

Huffman's algorithm is often used in algorithms textbooks as an example of a process that requires a dynamic priority queue. If a heap is used, for example, the n initial and $n-2$ subsequent insert operations, take a total of $O(n \log n)$ time, as do the $2(n-1)$ extract-min operations.

A simpler approach is to first sort the n weights into increasing order and then apply an $O(n)$-time algorithm due to van Leeuwen [10]. Two sorted lists are maintained: a static one of original weights, representing the leaves of the Huffman tree, and a dynamic queue of internal nodes that is initially empty, to which new internal nodes are appended as they are created. Each iteration compares front-of-list elements from the two lists and combines the two that have the least weight and then adds the new internal node at the tail of the queue. The algorithm stops when the queue contains only one node; it is the last item that was added and is the root of the Huffman tree.

If the input weights are provided in an array $w_i = A[i \mid 0 \leq i < n]$ of sorted integers, that array can be processed in situ into an output array $\ell_i = A[i]$ in $O(n)$ time by van Leeuwen's technique using an implementation described by Moffat and Katajainen [5]. Each array element takes on values that are, variously, input weight, internal node weight, parent pointer, and then, finally, codeword length. Algorithm 1 is taken from Moffat and Katajainen [5] and describes this process in detail. There are three phases of operation. In the first phase, in steps 2–2, leaf weights in $A[\mathit{leaf} \ldots n-1]$ are combined with a queue of internal node weights in $A[\mathit{root} \ldots \mathit{next}-1]$ to form a list of parent pointers in $A[0 \ldots \mathit{root}-1]$. At the end of this phase, $A[0 \ldots n-3]$ is a list of parents, $A[n-2]$ is the sum of the weights, and $A[n-1]$ is unused.

In phase 2 (steps 12–3), the set of parent pointers of internal nodes is converted to a set of internal node depths. This mapping is done by processing the tree from the root down, making the depth of each node one greater than the depth of its parent.

Algorithm 1 Compute Huffman codeword lengths

```
0: function $calc\_huff\_lens(A, n)$                      ▷ Input: $A[i-1] \le A[i]$ for $0 < i < n$
1:   // Phase 1
2:   set $leaf \leftarrow 0$ and $root \leftarrow 0$
3:   for $next \leftarrow 0$ to $n-2$ do
4:     if $leaf \ge n$ or ($root < next$ and $A[root] < A[leaf]$) then
5:       set $A[next] \leftarrow A[root]$ and $A[root] \leftarrow next$ and $root \leftarrow root + 1$      ▷ Use internal node
6:     else
7:       set $A[next] \leftarrow A[leaf]$ and $leaf \leftarrow leaf + 1$      ▷ Use leaf node
8:     end if
9:     repeat steps 1–8, but adding to $A[next]$ rather than assigning to it      ▷ Find second child
10:  end for
11:  // Phase 2
12:  set $A[n-2] \leftarrow 0$
13:  for $next \leftarrow n-3$ downto 0 do
14:    set $A[next] \leftarrow A[A[next]] + 1$      ▷ Compute depths of internal nodes
15:  end for
16:  // Phase 3
17:  set $avail \leftarrow 1$ and $used \leftarrow 0$ and $depth \leftarrow 0$ and $root \leftarrow n-2$ and $next \leftarrow n-1$
18:  while $avail > 0$ do
19:    while $root \ge 0$ and $A[root] = depth$ do
                                   ▷ Count internal nodes used at depth $depth$
20:      set $used \leftarrow used + 1$ and $root \leftarrow root - 1$
21:    end while
22:    while $avail > used$ do      ▷ Assign as leaves any nodes that are not internal
23:      set $A[next] \leftarrow d$ and $next \leftarrow next - 1$ and $avail \leftarrow avail - 1$
24:    end while
25:    set $avail \leftarrow 2 \cdot used$ and $depth \leftarrow depth + 1$ and $used \leftarrow 0$      ▷ Move to next depth
26:  end while
27:  return $A$      ▷ Output: $A[i]$ is the length $\ell_i$ of the $i$ th codeword
28: end function
```

Phase 3 (steps 17–4) then processes those internal node depths and converts them to a list of leaf depths. At each depth, some total number *avail* of nodes exist, being twice the number of internal nodes at the previous depth. Some number *used* of those are internal nodes; the balance must thus be leaf nodes at this depth and can be assigned as codeword lengths. Initially there is one node available at $depth = 0$, representing the root of the whole Huffman tree. Table 1 shows several snapshots of the Moffat and Katajainen code construction process when applied to the example sequence of weights.

Nonbinary Output Alphabets

The example Huffman tree developed in Fig. 1 and the process shown in Algorithm 1 assume that the output alphabet is binary. Huffman noted in his original paper that for r-ary alphabets all that is required is to add additional dummy symbols of weight zero, so as to bring the total number of symbols to be one more than a multiple of $(r-1)$. Each merging step then combines r leaf or internal nodes to form a new root node and decreases the number of items by $r-1$.

Dynamic Huffman Coding

Another assumption made by the processes described so far is that the symbol weights are known in advance and that the code that is computed can be *static*. This assumption can be satisfied, for example, by making a first pass over the message that is to be encoded. In a *dynamic* coding system, symbols must be coded on the fly, as soon as they are received by the encoder. To achieve this, the code must be adaptive, so that it can be altered after each symbol. Vitter [11] summarizes earlier work by Gallager [2], Knuth [4], and Cormack and Horspool [1] and describes a mechanism in which the total encoding cost, including the cost of keeping the code tree up to date, is

Huffman Coding, Table 1 Sequence of values computed by Algorithm 1 for the example weights. The first row shows the initial state of the array, with $A[i] = w_i$. Values "-2-" indicate parent pointers of internal nodes that have already been merged; italic values "*7*" indicate weights of internal nodes before being merged; values "(4)" indicate depths of internal nodes; bold values "**5**" indicate depths of leaves; and values "–" are unused

	i									
	0	1	2	3	4	5	6	7	8	9
Initial arrangement, $A[i] = w_i$	1	1	1	1	3	4	4	7	9	9
Phase 1, $root = 3, next = 5, leaf = 7$	-2-	-2-	-4-	*7*	*8*	–	–	7	9	9
Phase 1, finished, $root = 8$	-2-	-2-	-4-	-5-	-6-	-7-	-8-	-8-	*40*	–
Phase 2, $next = 4$	-2-	-2-	-4-	-5-	-6-	(2)	(1)	(1)	(0)	–
Phase 2, finished	(4)	(4)	(3)	(3)	(2)	(2)	(1)	(1)	(0)	–
Phase 3, $next = 5, avail = 4$	(4)	(4)	(3)	(3)	(2)	(2)	**3**	**3**	**2**	**2**
Final arrangement, $A[i] = \ell_i$	**5**	**5**	**5**	**5**	**4**	**4**	**3**	**3**	**2**	**2**

$O(1)$ per output bit. Turpin and Moffat [9] describe an alternative approximate algorithm that reduces the time required by a constant factor, by collecting the frequency updates into batches and allowing controlled inefficiency in the length of the coded output sequence. Their "GEO" Coding method is faster than dynamic Huffman Coding and also faster than dynamic Arithmetic Coding, which is comparable in speed to dynamic Huffman Coding, but uses less space for the dynamic frequency-counting data structure.

Applications

Minimum-redundancy codes have widespread use in data compression systems. The sequences of weights are usually conditioned according to a *model*, rather than taken as plain symbol frequency counts in the source message. The use of multiple conditioning contexts, and hence multiple codes, one per context, allows improved compression when symbols are not independent in the message, as is the case in natural language data. However, when the contexts are sufficiently specific that highly biased probability distributions arise, Arithmetic Coding will yield superior compression effectiveness.

Turpin and Moffat [8] consider several ancillary components of Huffman Coding, including methods for transmitting the description of the code to the decoder.

Cross-References

▸ Arithmetic Coding for Data Compression
▸ Compressing Integer Sequences

Recommended Reading

1. Cormack GV, Horspool RN (1984) Algorithms for adaptive Huffman codes. Inf Process Lett 18(3):159–165
2. Gallager RG (1978) Variations on a theme by Huffman. IEEE Trans Inf Theory IT-24(6):668–674
3. Huffman DA (1952) A method for the construction of minimum-redundancy codes. Proc Inst Radio Eng 40(9):1098–1101
4. Knuth DE (1985) Dynamic Huffman coding. J Algorithms 6(2):163–180
5. Moffat A, Katajainen J (1995) In-place calculation of minimum-redundancy codes. In: Proceedings of the Workshop on Algorithms and Data Structures, Kingston, pp 393–402
6. Moffat A, Turpin A (1997) On the implementation of minimum-redundancy prefix codes. IEEE Trans Commun 45(10):1200–1207
7. Stix G (1991) Profile: information theorist David A. Huffman. Sci Am 265(3):54–58. Reproduced at http://www.huffmancoding.com/my-uncle/david-bio. Accessed 15 July 2014
8. Turpin A, Moffat A (2000) Housekeeping for prefix coding. IEEE Trans Commun 48(4):622–628
9. Turpin A, Moffat A (2001) On-line adaptive canonical prefix coding with bounded compression loss. IEEE Trans Inf Theory 47(1):88–98
10. van Leeuwen J (1976) On the construction of Huffman trees. In: Proceedings of the International Conference on Automata, Languages, and Programming, Edinburgh University, Edinburgh, pp 382–410
11. Vitter JS (1987) Design and analysis of dynamic Huffman codes. J ACM 34(4):825–845