D

Data Migration

2004; Khuller, Kim, Wan

Yoo-Ah Kim
Computer Science and Engineering Department,
University of Connecticut, Storrs, CT, USA

Keywords and Synonyms

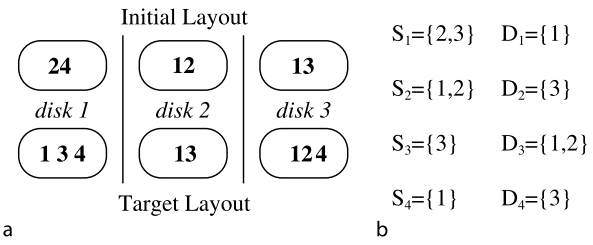
File transfers; Data movements

Problem Definition

The problem is motivated by the need to manage data on a set of storage devices to handle dynamically changing demand. To maximize utilization, the data layout (i. e., a mapping that specifies the subset of data items stored on each disk) needs to be computed based on disk capacities as well as the demand for data. Over time as the demand for data changes, the system needs to create new data layout. The data migration problem is to compute an efficient schedule for the set of disks to convert an initial layout to a target layout.

The problem is defined as follows. Suppose that there are N disks and Δ data items, and an initial layout and a target layout are given (see Fig. 1a for an example). For each item i, source disks S_i is defined to be a subset of disks which have item i in the initial layout. Destination disks D_i is a subset of disks that want to receive item i. In other words, disks in D_i have to store item i in the target layout but do not have to store it in the initial layout. Figure 1b shows the corresponding S_i and D_i. It is assumed that $S_i \neq \emptyset$ and $D_i \neq \emptyset$ for each item i. Data migration is the transfer of data to have all D_i receive data item i residing in S_i initially, and the goal is to minimize the total amount of time required for the transfers.

Assume that the underlying network is fully connected and the data items are all the same size. In other words, it takes the same amount of time to migrate an item from one disk to another. Therefore, migrations are performed in rounds. Consider the half-duplex model, where each disk can participate in the transfer of only one item – either as a sender or as a receiver. The objective is to find a migration schedule using the minimum number of rounds. No bypass nodes[1] can be used and therefore all data items are sent only to disks that desire them.



Data Migration, Figure 1
Left **An example of initial and target layout and** ***right*** **their corresponding S_i's and D_i's**

Key Results

Khuller et al. [11] developed a 9.5-approximation for the data migration problem, which was later improved to $6.5 + o(1)$. In the next subsection, the lower bounds of the problem are first examined.

Notations and Lower Bounds

1. **Maximum in-degree (β):** Let β_j be the number of data items that a disk j has to receive. In other words, $\beta_j = |\{i | j \in D_i\}|$. Then $\beta = \max_j \beta_j$ is a lower bound on the optimal as a disk can receive only one data item in one round.
2. **Maximum number of items that a disk may be a source or destination for (α):** For each item i, at least one disk in S_i should be used as a source for the item, and this disk is called a *primary source*. A unique primary source $s_i \in S_i$ for each item i that minimizes

[1] A bypass node is a node that is not the target of a move operation, but is used as an intermediate holding point for a data item.

$\alpha = \max_{j=1,\ldots,N}(|\{i|j = s_i\}| + \beta_j)$ can be found using a network flow. Note that $\alpha \geq \beta$, and α is also a lower bound on the optimal solution.

3. **Minimum time required for cloning (M):** Let a disk j make a copy of item i at the kth round. At the end of the mth round, the number of copies that can be created from the copy is at most 2^{m-k} as in each round the number of copies can only be doubled. Also note that each disk can make a copy of only one item in one round. Since at least $|D_i|$ copies of item i need to be created, the minimum m that satisfies the following linear program gives a lower bound on the optimal solution: **L(m):**

$$\sum_j \sum_{k=1}^{m} 2^{m-k} x_{ijk} \geq |D_i| \quad \text{for all } i \tag{1}$$

$$\sum_i x_{ijk} \leq 1 \quad \text{for all } j, k \tag{2}$$

$$0 \leq x_{ijk} \leq 1 \tag{3}$$

Data Migration Algorithm

A 9.5-approximation can be obtained as follows. The algorithm first computes representative sets for each item and sends the item to the representative sets first, which in turn send the item to the remaining set. Representative sets are computed differently depending on the size of D_i.

Representatives for Big Sets For sets with size at least β, a *disjoint* collection of *representative sets* R_i, $i = 1 \ldots \Delta$ has to satisfy the following properties: Each R_i should be a subset of D_i and $|R_i| = \lfloor |D_i|/\beta \rfloor$. The representative sets can be found using a network flow.

Representatives for Small Sets For each item i, let $\gamma_i = |D_i| \bmod k$. A *secondary representative* r_i in D_i for the items with $\gamma_i \neq 0$ needs to be computed. A disk j can be a secondary representative r_i for several items as long as $\sum_{i \in I_j} \gamma_i \leq 2\beta - 1$, where I_j is a set of items for which j is a secondary representative. This can be done by applying the Shmoys–Tardos algorithm [17] for the generalized assignment problem.

Scheduling Migrations Given representatives for all data items, migrations can be done in three steps as follows:

1. **Migration to R_i:** Each item i is first sent to the set R_i. By converting a fractional solution given in $L(M)$, one can find a migration schedule from s_i to R_i and it requires at most $2M + \alpha$ rounds.
2. **Migration to r_i:** Item i is sent from primary source s_i to r_i. The migrations can be done in 1.5α rounds, using an algorithm for edge coloring [16].
3. **Migration to the remaining disks:** A transfer graph from representatives to the remaining disks can now be created as follows. For each item i, add directed edges from disks in R_i to $(\beta - 1)\lfloor \frac{|D_i|}{\beta} \rfloor$ disks in $D_i \setminus R_i$ such that the out-degree of each node in R_i is at most $\beta - 1$ and the in-degree of each node in $D_i \setminus R_i$ from R_i is 1. A directed edge is also added from the secondary representative r_i of item i to the remaining disks in D_i which do not have an edge coming from R_i. It has been shown that the maximum degree of the transfer graph is at most $4\beta - 5$ and the multiplicity is $\beta + 2$. Therefore, migration for the transfer graph can be done in $5\beta - 3$ rounds using an algorithm for multigraph edge coloring [18].

Analysis Note that the total number of rounds required in the algorithm described in "Data Migration Algorithm" is at most $2M + 2.5\alpha + 5\beta - 3$. As α, β and M are lower bounds on the optimal number of rounds, the abovementioned algorithm gives a 9.5-approximation.

Theorem 1 ([11]) *There is a 9.5-approximation algorithm for the data migration problem.*

Khuller et al. [10] later improved the algorithm and obtained a $(6.5 + o(1))$-approximation.

Theorem 2 ([10]) *There is a $(6.5 + o(1))$-approximation algorithm for the data migration problem.*

Applications

Data Migration in Storage Systems

Typically, a large storage server consists of several disks connected using a dedicated network, called a *storage area network*. To handle high demand, especially for multimedia data, a common approach is to replicate data objects within the storage system. Disks typically have constraints on storage as well as the number of clients that can access data from a single disk simultaneously. Approximation algorithms have been developed to map known demand for data to a specific data layout pattern to maximize utilization[2] [4,8,14,15]. In the layout, they compute not only how many copies of each item need to be created, but also a layout pattern that specifies the precise subset of items on each disk. The problem is NP-hard, but there are polynomial-time approximation schemes [4,8,14]. Given

[2]The utilization is the total number of clients that can be assigned to a disk that contains the data they want.

the relative demand for data, the algorithm computes an almost optimal layout.

Over time as the demand for data changes, the system needs to create new data layouts. To handle high demand for popular objects, new copies may have to be dynamically created and stored on different disks. The data migration problem is to compute a specific schedule for the set of disks to convert an initial layout to a target layout. Migration should be done as quickly as possible since the performance of the system will be suboptimal during migration.

Gossiping and Broadcasting

The data migration problem can be considered as a generalization of gossiping and broadcasting. The problems of gossiping and broadcasting play an important role in the design of communication protocols in various kinds of networks and have been extensively studied (see for example [6,7] and the references therein). The gossip problem is defined as follows. There are n individuals and each individual has an item of gossip that he/she wish to communicate to everyone else. Communication is typically done in rounds, where in each round an individual may communicate with at most one other individual. Some communication models allow for the full exchange of all items of gossip known to each individual in a single round. In addition, there may be a communication graph whose edge indicates which pairs of individuals are allowed to communicate directly in each round. In the broadcast problem, one individual needs to convey an item of gossip to every other individual. The data migration problem generalizes the gossiping and broadcasting in three ways: (1) each item of gossip needs to be communicated to only a subset of individuals; (2) several items of gossip may be known to an individual; (3) a single item of gossip can initially be shared by several individuals.

Open Problems

The data migration problem is NP-hard by reduction from the edge coloring problem. However, no inapproximability results are known for the problem. As the current best approximation factor is relatively high $(6.5 + o(1))$, it is an interesting open problem to narrow the gap between the approximation guarantee and the inapproximability.

Another open problem is to combine data placement and migration problems. This question was studied by Khuller et al. [9]. Given the initial layout and the new demand pattern, their goal was to find a set of data migrations that can be performed in a specific number of rounds and gives the best possible layout to the current demand pattern. They showed that even one-round migration is NP-hard and presented a heuristic algorithm for the one-round migration problem. The experiments showed that performing a few rounds of one-round migration consecutively works well in practice. Obtaining nontrivial approximation algorithms for this problem would be interesting future work.

Data migration in a heterogeneous storage system is another interesting direction for future research. Most research on data migration has focused mainly on homogeneous storage systems, assuming that disks have the same fixed capabilities and the network connections are of the same fixed bandwidth. In practice, however, large-scale storage systems may be heterogenous. For instance, disks tend to have heterogeneous capabilities as they are added over time owing to increasing demand for storage capacity. Lu et al. [13] studied the case when disks have variable bandwidth owing to the loads on different disks. They used a control-theoretic approach to generate adaptive rates of data migrations which minimize the degradation of the quality of the service. The algorithm reduces the latency experienced by clients significantly compared with the previous schemes. However, no theoretical bounds on the efficiency of data migrations were provided. Coffman et al. [2] studied the case when each disk i can handle p_i transfers simultaneously and provided approximation algorithms. Some papers [2,12] considered the case when the lengths of data items are heterogenous (but the system is homogeneous), and present approximation algorithms for the problem.

Experimental Results

Golubchik et al. [3] conducted an extensive study of the performance of data migration algorithms under different changes in user-access patterns. They compared the 9.5-approximation [11] and several other heuristic algorithms. Some of these heuristic algorithms cannot provide constant approximation guarantees, while for some of the algorithms no approximation guarantees are known. Although the worst-case performance of the algorithm by Khuller et al. [11] is 9.5, in the experiments the number of rounds required was less than 3.25 times the lower bound.

They also introduced the *correspondence problem*, in which a matching between disks in the initial layout with disks in the target layout is computed so as to minimize changes. A good solution to the correspondence problem can improve the performance of the data migration algorithms by a factor of 4.4 in their experiments, relative to a bad solution.

URL to Code

http://www.cs.umd.edu/projects/smart/data-migration/

Cross References

- ► Broadcasting in Geometric Radio Networks
- ► Deterministic Broadcasting in Radio Networks

Recommended Reading

A special case of the data migration problem was studied by Anderson et al. [1] and Hall et al. [5]. They assumed that a data transfer graph is given, in which a node corresponds to each disk and a directed edge corresponds to each move operation that is specified (the creation of new copies of data items is not allowed). Computing a data movement schedule is exactly the problem of edge-coloring the transfer graph. Algorithms for edge-coloring multigraphs can now be applied to produce a migration schedule since each color class represents a matching in the graph that can be scheduled simultaneously. Computing a solution with the minimum number of rounds is NP-hard, but several good approximation algorithms are available for edge coloring. With space constraints on the disk, the problem becomes more challenging. Hall et al. [5] showed that with the assumption that each disk has one spare unit of storage, very good constant factor approximations can be developed. The algorithms use at most $4\lceil\Delta/4\rceil$ colors with at most $n/3$ bypass nodes, or at most $6\lceil\Delta/4\rceil$ colors without bypass nodes.

Most of the results on the data migration problem deal with the half-duplex model. Another interesting communication model is the full-duplex model where each disk can act as a sender and a receiver in each round for a single item. There is a $(4 + o(1))$-approximation algorithm for the full-duplex model [10].

1. Anderson, E., Hall, J., Hartline, J., Hobbes, M., Karlin, A., Saia, J., Swaminathan, R., Wilkes, J.: An experimental study of data migration algorithms. In: Workshop on Algorithm Engineering (2001)
2. Coffman, E., Garey, M., Jr., Johnson, D., Lapaugh, A.: Scheduling file transfers. SIAM J. Comput. **14**(3), 744–780 (1985)
3. Golubchik, L., Khuller, S., Kim, Y., Shargorodskaya, S., Wan., Y.: Data migration on parallel disks. In: 12th Annual European Symposium on Algorithms (ESA) (2004)
4. Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., Zhu, A.: Approximation algorithms for data placement on parallel disks. In: Symposium on Discrete Algorithms, pp. 223–232. Society for Industrial and Applied Mathematics, Philadelphia (2000)
5. Hall, J., Hartline, J., Karlin, A., Saia, J., Wilkes, J.: On algorithms for efficient data migration. In: SODA, pp. 620–629. Society for Industrial and Applied Mathematics, Philadelphia (2001)
6. Hedetniemi, S.M., Hedetniemi, S.T., Liestman, A.: A survey of gossiping and broadcasting in communication networks. Networks **18**, 129–134 (1988)
7. Hromkovic, J., Klasing, R., Monien, B., Peine, R.: Dissemination of information in interconnection networks (broadcasting and gossiping). In: Du, D.Z., Hsu, F. (eds.) Combinatorial Network Theory, pp. 125–212. Kluwer Academic Publishers, Dordrecht (1996)
8. Kashyap, S., Khuller, S.: Algorithms for non-uniform size data placement on parallel disks. In: Conference on FST&TCS Conference. LNCS, vol. 2914, pp. 265–276. Springer, Heidelberg (2003)
9. Kashyap, S., Khuller, S., Wan, Y-C., Golubchik, L.: Fast reconfiguration of data placement in parallel disks. In: Workshop on Algorithm Engineering and Experiments (2006)
10. Khuller, S., Kim, Y., Malekian, A.: Improved algorithms for data migration. In: 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (2006)
11. Khuller, S., Kim, Y., Wan, Y.-C.: Algorithms for data migration with cloning. SIAM J. Comput. **33**(2), 448–461 (2004)
12. Yoo-Ah Kim. Data migration to minimize the average completion time. J. Algorithms **55**, 42–57 (2005)
13. Lu, C., Alvarez, G.A., Wilkes, J.: Aqueduct:online data migration with performance guarantees. In: Proceedings of the Conference on File and Storage Technologies (2002)
14. Shachnai, H., Tamir, T.: Polynomial time approximation schemes for class-constrained packing problems. J. Sched. **4**(6) 313–338 (2001)
15. Shachnai, H., Tamir, T.: On two class-constrained versions of the multiple knapsack problem. Algorithmica **29**(3), 442–467 (2001)
16. Shannon, C.E.: A theorem on colouring lines of a network. J. Math. Phys. **28**, 148–151 (1949)
17. Shmoys, D.B., Tardos, E.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**(3), 461–474 (1993)
18. Vizing, V.G.: On an estimate of the chromatic class of a p-graph (Russian). Diskret. Analiz. **3**, 25–30 (1964)

Data Reduction for Domination in Graphs

2004; Alber, Fellows, Niedermeier

ROLF NIEDERMEIER
Department of Math and Computer Science, University of Jena, Jena, Germany

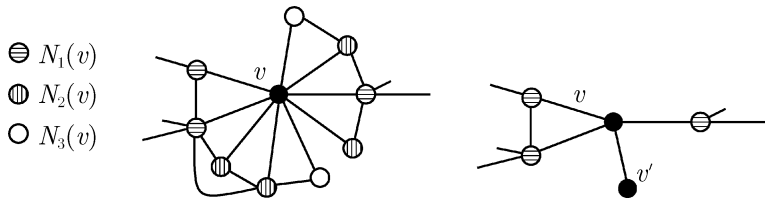
Keywords and Synonyms

Dominating set; Reduction to a problem kernel; Kernelization

Problem Definition

The NP-complete DOMINATING SET problem is a notoriously hard problem:

Problem 1 (Dominating Set)
INPUT: *An undirected graph* $G = (V, E)$ *and an integer* $k \geq 0$.



Data Reduction for Domination in Graphs, Figure 1
The *left-hand side* shows the partitioning of the neighborhood of a single vertex *v*. The *right-hand side* shows the result of applying the presented data reduction rule to this particular (sub)graph

QUESTION: *Is there an $S \subseteq V$ with $|S| \leq k$ such that every vertex $v \in V$ is contained in S or has at least one neighbor in S?*

For instance, for an n-vertex graph its optimization version is known to be polynomial-time approximable only up to a factor of $\Theta(\log n)$ unless some standard complexity-theoretic assumptions fail [9]. In terms of parametrized complexity, the problem is shown to be W[2]-complete [8]. Although still NP-complete when restricted to planar graphs, the situation much improves here. In her seminal work, Baker showed that there is an efficient polynomial-time approximation scheme (PTAS) [6], and the problem also becomes fixed-parameter tractable [2,4] when restricted to planar graphs. In particular, the problem becomes accessible to fairly effective data reduction rules and a kernelization result (see [16] for a general description of data reduction and kernelization) can be proven. This is the subject of this entry.

Key Results

The key idea behind the data reduction is preprocessing based on locally acting simplification rules. Exemplary, here we describe a rule where the local neighborhood of each graph vertex is considered. To this end, we need the following definitions.

We partition the neighborhood $N(v)$ of an arbitrary vertex $v \in V$ in the input graph into three disjoint sets $N_1(v)$, $N_2(v)$, and $N_3(v)$ depending on local neighborhood structure. More specifically, we define

- $N_1(v)$ to contain all neighbors of v that have edges to vertices that are not neighbors of v;
- $N_2(v)$ to contain all vertices from $N(v) \setminus N_1(v)$ that have edges to at least one vertex from $N_1(v)$;
- $N_3(v)$ to contain all neighbors of v that are neither in $N_1(v)$ nor in $N_2(v)$.

An example which illustrates such a partitioning is given in Fig. 1 (left-hand side). A helpful and intuitive interpretation of the partition is to see vertices in $N_1(v)$ as *exits* because they have direct connections to the world outside the closed neighborhood of v, vertices in $N_2(v)$ as *guards* because they have direct connections to exits, and vertices in $N_3(v)$ as *prisoners* because they do not see the world outside $\{v\} \cup N(v)$.

Now consider a vertex $w \in N_3(v)$. Such a vertex only has neighbors in $\{v\} \cup N_2(v) \cup N_3(v)$. Hence, to dominate w, at least one vertex of $\{v\} \cup N_2(v) \cup N_3(v)$ *must* be contained in a dominating set for the input graph. Since v can dominate all vertices that would be dominated by choosing a vertex from $N_2(v) \cup N_3(v)$ into the dominating set, we obtain the following data reduction rule.

If $N_3(v) \neq \emptyset$ for some vertex v, then remove $N_2(v)$ and $N_3(v)$ from G
and add a new vertex v'
with the edge $\{v, v'\}$ to G.

Note that the new vertex v' can be considered as a "gadget vertex" that "enforces" v to be chosen into the dominating set. It is not hard to verify the correctness of this rule, that is, the original graph has a dominating set of size k iff the reduced graph has a dominating set of size k. Clearly, the data reduction can be executed in polynomial time [5]. Note, however, that there are particular "diamond" structures that are not amenable to this reduction rule. Hence, a second, somewhat more complicated rule based on considering the joint neighborhood of *two* vertices has been introduced [5].

Altogether, the following core result could be shown [5].

Theorem 1 *A planar graph $G = (V, E)$ can be reduced in polynomial time to a planar graph $G' = (V', E')$ such that G has a dominating set of size k iff G' has a dominating set of size k and $|V'| = O(k)$.*

In other words, the theorem states that the DOMINATING SET in planar graphs has a linear-size problem kernel. The upper bound on $|V'|$ was first shown to be $335k$ [5] and

was then further improved to $67k$ [7]. Moreover, the results can be extended to graphs of bounded genus [10]. In addition, similar results (linear kernelization) have been recently obtained for the FULL-DEGREE SPANNING TREE problem in planar graphs [13]. Very recently, these results have been generalized into a methodological framework [12].

Applications

DOMINATING SET is considered to be one of the most central graph problems [14,15]. Its applications range from facility location to bioinformatics.

Open Problems

The best lower bound for the size of a problem kernel for DOMINATING SET in planar graphs is $2k$ [7]. Thus, there is quite a gap between known upper and lower bounds. In addition, there have been some considerations concerning a generalization of the above-discussed data reduction rules [3]. To what extent such extensions are of practical use remains to be explored. Finally, a study of deeper connections between Baker's PTAS results [6] and linear kernelization results for DOMINATING SET in planar graphs seems to be worthwhile for future research. Links concerning the class of problems amenable to both approaches have been detected recently [12]. The research field of data reduction and problem kernelization as a whole together with its challenges is discussed in a recent survey [11].

Experimental Results

The above-described theoretical work has been accompanied by experimental investigations on synthetic as well as real-world data [1]. The results have been encouraging in general. However, note that grid structures seem to be a hard case where the data reduction rules remained largely ineffective.

Cross References

► Connected Dominating Set

Recommended Reading

1. Alber, J., Betzler, N., Niedermeier, R.: Experiments on data reduction for optimal domination in networks. Ann. Oper. Res. **146**(1), 105–117 (2006)
2. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for Dominating Set and related problems on planar graphs. Algorithmica **33**(4), 461–493 (2002)
3. Alber, J., Dorn, B., Niedermeier, R.: A general data reduction scheme for domination in graphs. In: Proc. 32nd SOFSEM. LNCS, vol. 3831, pp. 137–147. Springer, Berlin (2006)
4. Alber, J., Fan, H., Fellows, M.R., Fernau, H., Niedermeier, R., Rosamond, F., Stege, U.: A refined search tree technique for Dominating Set on planar graphs. J. Comput. Syst. Sci. **71**(4), 385–405 (2005)
5. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial time data reduction for Dominating Set. J. ACM **51**(3), 363–384 (2004)
6. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM **41**(1), 153–180 (1994)
7. Chen, J., Fernau, H., Kanj, I.A., Xia, G.: Parametric duality and kernelization: lower bounds and upper bounds on kernel size. SIAM J. Comput. **37**(4), 1077–1106 (2007)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
9. Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM **45**(4), 634–652 (1998)
10. Fomin, F.V., Thilikos, D.M.: Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In: Proc. 31st ICALP. LNCS, vol. 3142, pp. 581–592. Springer, Berlin (2004)
11. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News **38**(1), 31–45 (2007)
12. Guo, J., Niedermeier, R.: Linear problem kernels for NP-hard problems on planar graphs. In: Proc. 34th ICALP. LNCS, vol. 4596, pp. 375–386. Springer, Berlin (2007)
13. Guo, J., Niedermeier, R., Wernicke, S.: Fixed-parameter tractability results for full-degree spanning tree and its dual. In: Proc. 2nd IWPEC. LNCS, vol. 4196, pp. 203–214. Springer, Berlin (2006)
14. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Domination in Graphs: Advanced Topics. Pure and Applied Mathematics, vol. 209. Marcel Dekker, New York (1998)
15. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of Domination in Graphs. Pure and Applied Mathematics, vol. 208. Marcel Dekker, New York (1998)
16. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, New York (2006)

Decoding

► Decoding Reed–Solomon Codes
► List Decoding near Capacity: Folded RS Codes

Decoding Reed–Solomon Codes

1999; Guruswami, Sudan

VENKATESAN GURUSWAMI
Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Keywords and Synonyms

Decoding; Error correction

Problem Definition

In order to ensure the integrity of data in the presence of errors, an *error-correcting code* is used to *encode* data into a redundant form (called a *codeword*). It is natural to view both the original data (or *message*) as well as the associated codeword as strings over a finite alphabet. Therefore, an error-correcting code C is defined by an injective encoding map $E : \Sigma^k \to \Sigma^n$, where k is called the *message length*, and n the *block length*. The codeword, being a redundant form of the message, will be longer than the message. The *rate* of an error-correcting code is defined as the ratio k/n of the length of the message to the length of the codeword. The rate is a quantity in the interval $(0, 1]$, and is a measure of the redundancy introduced by the code. Let $R(C)$ denote the rate of a code C.

The redundancy built into a codeword enables detection and hopefully also correction of any errors introduced, since only a small fraction of all possible strings will be legitimate codewords. Ideally, the codewords encoding different messages should be "far-off" from each other, so that one can recover the original codeword even when it is distorted by moderate levels of noise. A natural measure of distance between strings is the Hamming distance. The Hamming distance between strings $x, y \in \Sigma^*$ of the same length, denoted $\mathsf{dist}(x, y)$, is defined to be the number of positions i for which $x_i \neq y_i$.

The *minimum distance*, or simply *distance*, of an error-correcting code C, denoted $d(C)$, is defined to be the smallest Hamming distance between the encodings of two distinct messages. The *relative distance* of a code C of block length n, denoted $\delta(C)$, is the ratio between its distance and n. Note that arbitrary corruption of any $\lfloor (d(C) - 1)/2 \rfloor$ of locations of a codeword of C cannot take it closer (in Hamming distance) to any other codeword of C. Thus in principle (i. e., efficiency considerations apart) error patterns of at most $\lfloor (d(C) - 1)/2 \rfloor$ errors can be corrected. This task is called *unique decoding* or *decoding up to half-the-distance*. Of course, it is also possible, and will often be the case, that error patterns with more than $d(C)/2$ errors can also be corrected by decoding the string to the closest codeword in Hamming distance. The latter task is called *Nearest-Codeword decoding* or *Maximum Likelihood Decoding (MLD)*.

One of the fundamental trade-offs in the theory of error-correcting codes, and in fact one could say all of combinatorics, is the one between rate $R(C)$ and distance $d(C)$ of a code. Naturally, as one increases the rate and thus number of codewords in a code, some two codewords must come closer together thereby lowering the distance. More qualitatively, this represents the tension between the redundancy of a code and its error-resilience. To correct more errors requires greater redundancy, and thus lower rate.

A code defined by encoding map $E : \Sigma^k \to \Sigma^n$ with minimum distance d is said to be an (n, k, d) code. Since there are $|\Sigma|^k$ codewords and only $|\Sigma^{k-1}|$ possible projections onto the first $k = 1$ coordinates, some two codewords must agree on the first $k - 1$ positions, implying that the distance d of the code must obey $d \leq n - k + 1$ (this is called the *Singleton bound*). Quite surprisingly, over large alphabets Σ there are well-known codes called Reed–Solomon codes which meet this bound exactly and have the optimal distance $d = n - k + 1$ for any given rate k/n. (In contrast, for small alphabets, such as $\Sigma = \{0, 1\}$, the optimal trade-off between rate and relative distance for an asymptotic family of codes is unknown and is a major open question in combinatorics.)

This article will describe the best known algorithmic results for error-correction of Reed–Solomon codes. These are of central theoretical and practical interest given the above-mentioned optimal trade-off achieved by Reed–Solomon codes, and their ubiquitous use in our everyday lives ranging from compact disc players to deep-space communication.

Reed–Solomon Codes

Definition 1 A *Reed–Solomon code* (or RS code), $\mathsf{RS}_{\mathbb{F},S}[n, k]$, is parametrized by integers n, k satisfying $1 \leq k \leq n$, a finite field $\mathbb{F}$ of size at least n, and a tuple $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n *distinct* elements from $\mathbb{F}$. The code is described as a subset of $\mathbb{F}^n$ as:

$$\mathsf{RS}_{\mathbb{F},S}[n, k] = \{(p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n)) | p(X) \in \mathbb{F}[X] \text{ is a polynomial of degree} \leq k - 1\}.$$

In other words, the message is viewed as a polynomial, and it is encoded by evaluating the polynomial at n distinct field elements $\alpha_1, \ldots, \alpha_n$. The resulting code is linear of dimension k, and its minimum distance equals $n - k + 1$, which matches the Singleton bound.

The distance property of RS codes follows from the fact that the evaluations of two distinct polynomials of degree less than k can agree on at most $k - 1$ field elements. Note that in the absence of errors, given a codeword $\mathbf{y} \in \mathbb{F}^n$, one can recover its corresponding message by polynomial interpolation on any k out of the n codeword positions. In fact, this also gives an *erasure decoding* algorithm when all but the information-theoretically bare minimum necessary k symbols are erased from the codeword (but the

receiver *knows* which symbols have been erased and the correct values of the rest of the symbols). The RS decoding problem, therefore, amounts to a noisy polynomial interpolation problem when some of the evaluation values are incorrect.

The holy grail in decoding RS codes would be to find the polynomial $p(X)$ whose RS encoding is closest in Hamming distance to a noisy string $\mathbf{y} \in \mathbb{F}^n$. One could then decode $\mathbf{y}$ to this message $p(X)$ as the maximum likelihood choice. No efficient algorithm for such nearest-codeword decoding is known for RS codes (or for that matter any family of "good" or non-trivial codes), and it is believed that the problem is NP-hard. Guruswami and Vardy [6] proved the problem to NP-hard over exponentially large fields, but this is a weak negative result since normally one considers Reed–Solomon codes over fields of size at most $O(n)$.

Given the intractability of nearest-codeword decoding in its extreme generality, lot of attention has been devoted to the *bounded distance decoding* problem, where one assumes that the string $\mathbf{y} \in \mathbb{F}^n$ to be decoded has at most e errors, and the goal is to find the Reed–Solomon codeword(s) within Hamming distance e from $\mathbf{y}$.

When $e < (n-k)/2$, this corresponds to decoding up to half the distance. This is a classical problem for which a polynomial time algorithm was first given by Peterson [8]. (It is notable that this even before the notion of polynomial time was put forth as the metric of theoretical efficiency.) The focus of this article is on a *list decoding* algorithm for Reed–Solomon codes due to Guruswami and Sudan [5] that decode beyond half the minimum distance. The formal problem and the key results are stated next.

Key Results

In this section, the main result of focus concerning decoding Reed–Solomon codes is stated. Given the target of decoding errors beyond half-the-minimum distance, one needs to deal with inputs where there may be more than one codeword within the radius e specified in the bounded distance decoding problem. This is achieved by a relaxation of decoding called *list decoding* where the decoder outputs a list of all codewords (or the corresponding messages) within Hamming distance e from the received word. If one wishes, one can choose the closest codeword among the list as the "most likely" answer, but there are many applications of Reed–Solomon decoding, for example to decoding concatenated codes and several applications in complexity theory and cryptography, where having the entire list of codewords adds to the power of the decoding primitive. The main result of Guruswami and Sudan [5], building upon the work of Sudan [9], is the following:

Theorem 1 ([5]) *Let $C = \mathsf{RS}_{\mathbb{F},S}[n,k]$ be a Reed–Solomon code over a field $\mathbb{F}$ of size $q \geq n$ with $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. There is a deterministic algorithm running in time polynomial in q that on input $y \in \mathbb{F}_q^n$ outputs a list of all polynomials $p(X) \in \mathbb{F}[X]$ of degree less than k for which $p(\alpha_i) \neq y_i$ for less than $n - \sqrt{(k-1)n}$ positions $i \in \{1, 2, \ldots, n\}$. Further, at most $O(n^2)$ polynomials will be output by the algorithm in the worst-case.*

Alternatively, one can correct a RS code of block length n and rate $R = k/n$ up to $n - \sqrt{(k-1)}$ errors, or equivalently a fraction $1 - \sqrt{R}$ of errors.

The Reed–Solomon decoding algorithm is based on the solution to the following more general polynomial reconstruction problem which seems like a natural algebraic question in itself. (The problem is more general than RS decoding since the α_i's need not be distinct.)

Problem 1 (Polynomial Reconstruction)
Input: *Integers $k, t \leq n$ and n distinct pairs $\{(\alpha_i, y_i)\}_{i=1}^n$ where $\alpha_i, y_i \in \mathbb{F}$.*
Output: *A list of all polynomials $p(X) \in \mathbb{F}[X]$ of degree less than k which satisfy $p(\alpha_i) = y_i$ for at least t values of $i \in [n]$.*

Theorem 2 *The polynomial reconstruction problem can be solved in time polynomial in $n, |\mathbb{F}|$, provided $t > \sqrt{(k-1)n}$.*

The reader is referred to the original papers [5,9], or a recent survey [1], for details on the above algorithm. A quick, high level peek into the main ideas is given below. The first step in the algorithm consists of an interpolation step where a nonzero bivariate polynomial $Q(X,Y)$ is "fit" through the n pairs (α_i, y_i), so that $Q(\alpha_i, y_i) = 0$ for every i. The key is to do this with relatively low degree; in particular one can find such a $Q(X,Y)$ with so-called $(1, k-1)$-weighted degree at most $D \approx \sqrt{2(k-1)n}$. This degree budget on Q implies that for any polynomial $p(X)$ of degree less than k, $Q(X, p(X))$ will have degree at most D. Now whenever $p(\alpha_i) = y_i$, $Q(\alpha_i, p(\alpha)i)) = Q(\alpha_i, y_i) = 0$. Therefore, if a polynomial $p(X)$ satisfies $p(\alpha_i) = y_i$ for at least t values of i, then $Q(X, p(X))$ has at least t roots. On the other hand the polynomial $Q(X, p(X))$ has degree at most D. Therefore, if $t > D$, one must have $Q(X, p(X)) = 0$, or in other words $Y - p(X)$ is a factor of $Q(X,Y)$. The second step of the algorithm factorized the polynomial $Q(X,Y)$, and all polynomials $p(X)$ that must be output will be found as factors $Y - p(X)$ of $Q(X,Y)$.

Note that since $D \approx \sqrt{2(k-1)n}$ this gives an algorithm for polynomial reconstruction provided the agreement parameter t satisfies $t > \sqrt{2(k-1)n}$ [9]. To get an algorithm for $t > \sqrt{(k-1)n}$, and thus decode beyond half the minimum distance $(n-k)/2$ for all parameter choices for k, n, Guruswami and Sudan [5] use the crucial idea of allowing "multiple roots" in the interpolation step. Specifically, the polynomial Q is required to have $r \geq 1$ roots at each pair (α_i, y_i) for some integer multiplicity parameter r (the notion needs to be formalized properly, see [5] for details). This necessitates an increase in the $(1, k-1)$-weighted degree of a factor of about $r/\sqrt{2}$, but the gain is that one gets a factor r more roots for the polynomial $Q(X, p(X))$. These facts together lead to an algorithm that works as long as $t > \sqrt{(k-1)n}$.

There is an additional significant benefit offered by the multiplicity based decoder. The multiplicities of the interpolation points need not all be equal and they can picked in proportion to the reliability of different received symbols. This gives a powerful way to exploit "soft" information in the decoding stage, leading to impressive coding gains in practice. The reader is referred to the paper by Koetter and Vardy [7] for further details on using multiplicities to encode symbol level reliability information from the channel.

Applications

Reed–Solomon codes have been extensively studied and are widely used in practice. The above decoding algorithm corrects more errors beyond the traditional half the distance limit and therefore directly advances the state of the art on this important algorithmic task. The RS list decoding algorithm has also been the backbone for many further developments in algorithmic coding theory. In particular, using this algorithm in concatenation schemes leads to good binary list-decodable codes. A variant of RS codes called folded RS codes have been used to achieve the optimal trade-off between error-correction radius and rate [3] (see the companion encyclopedia entry by Rudra on folded RS codes).

The RS list decoding algorithm has also found many surprising applications beyond coding theory. In particular, it plays a key role in several results in cryptography and complexity theory (such as constructions of randomness extractors and pseudorandom generators, hardness amplification, constructions to hardcore predicates, traitor tracing, reductions connecting worst-case hardness to average-case hardness, etc.); more information can be found, for instance, in [10] or Chap. 12 in [2].

Open Problems

The most natural open question is whether one can improve the algorithm further and correct more than a fraction $1 - \sqrt{R}$ of errors for RS codes of rate R. It is important to note that there is a combinatorial limitation to the number of errors one can list decode from. One can only list decode in polynomial time from a fraction ρ of errors if for every received word $\mathbf{y}$ the number of RS codewords within distance $e = \rho n$ of $\mathbf{y}$ is bounded by a polynomial function of the block length n. The largest ρ for which this holds as a function of the rate R is called the list decoding radius $\rho_{\mathsf{LD}} = \rho_{\mathsf{LD}}(R)$ of RS codes. The RS list decoding algorithm discussed here implies that $\rho_{\mathsf{LD}}(R) \geq 1 - \sqrt{R}$, and it is trivial to see than $\rho_{\mathsf{LD}}(R) \leq 1 - R$. Are there RS codes (perhaps based on specially structured evaluation points) for which $\rho_{\mathsf{LD}}(R) > 1 - \sqrt{R}$? Are there RS codes for which the $1 - \sqrt{R}$ radius (the so-called "Johnson bound") is actually tight for list decoding? For the more general polynomial reconstruction problem the $\sqrt{(k-1)n}$ agreement cannot be improved upon [4], but this is not known for RS list decoding.

Improving the NP-hardness result of [6] to hold for RS codes over polynomial sized fields and for smaller decoding radii remains an important challenge.

Cross References

- ► Learning Heavy Fourier Coefficients of Boolean Functions
- ► List Decoding near Capacity: Folded RS Codes
- ► LP Decoding

Recommended Reading

1. Guruswami, V.: Algorithmic Results in List Decoding. In: Foundations and Trends in Theoretical Computer Science, vol. 2, issue 2, NOW publishers, Hanover (2007)
2. Guruswami, V.: List Decoding of Error-Correcting Codes. Lecture Notes in Computer Science, vol. 3282. Springer, Berlin (2004)
3. Guruswami, V., Rudra, A.: Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. IEEE Trans. Inform. Theor. **54**(1), 135–150 (2008)
4. Guruswami, V., Rudra, A.: Limits to list decoding Reed–Solomon codes. IEEE Trans. Inf. Theory. **52**(8), 3642–3649 (2006)
5. Guruswami, V., Sudan, M.: Improved decoding of Reed–Solomon and algebraic-geometric codes. IEEE Trans. Inf. Theory. **45**(6), 1757–1767 (1999)
6. Guruswami, V., Vardy A.: Maximum Likelihood Decoding of Reed–Solomon codes is NP-hard. IEEE Trans. Inf. Theory. **51**(7), 2249–2256 (2005)
7. Koetter, R., Vardy, A.: Algebraic soft-decision decoding of Reed–Solomon codes. IEEE Trans. Inf. Theory. **49**(11), 2809–2825 (2003)

8. Peterson, W.W.: Encoding and error-correction procedures for Bose-Chaudhuri codes. IEEE Trans. Inf. Theory. **6**, 459–470 (1960)
9. Sudan, M.: Decoding of Reed–Solomon codes beyond the error-correction bound. J. Complex. **13**(1), 180–193 (1997)
10. Sudan, M.: List decoding: Algorithms and applications. SIGACT News. **31**(1), 16–27 (2000)

Decremental All-Pairs Shortest Paths

2004; Demetrescu, Italiano

Camil Demetrescu, Giuseppe F. Italiano
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Deletions-only dynamic all-pairs shortest paths

Problem Definition

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

This entry addressed the *decremental* version of the all-pairs shortest paths problem (APSP), which consists of maintaining a directed graph with real-valued edge weights under an intermixed sequence of the following operations:

`delete`(u, v): delete edge (u, v) from the graph.
`distance`(x, y): return the distance from vertex x to vertex y.
`path`(x, y): report a shortest path from vertex x to vertex y, if any.

A natural variant of this problem supports a generalized delete operation that removes a vertex and all edges incident to it. The algorithms addressed in this entry can deal with this generalized operation within the same bounds.

History of the Problem

A simple-minded solution to this problem would be to rebuild shortest paths from scratch after each deletion using the best static APSP algorithm so that distance and path queries can be reported in optimal time. The fastest known static APSP algorithm for arbitrary real weights has a running time of $O(mn + n^2 \log\log n)$, where m is the number of edges and n is the number of vertices in the graph [13]. This is $\Omega(n^3)$ in the worst case. Fredman [6] and later Takaoka [19] showed how to break this cubic barrier: the best asymptotic bound is by Takaoka, who showed how to solve APSP in $O(n^3 \sqrt{\log\log n / \log n})$ time.

Another simple-minded solution would be to answer queries by running a point-to-point shortest paths computation, without the need to update shortest paths at each deletion. This can be done with Dijkstra's algorithm [3] in $O(m + n \log n)$ time using the Fibonacci heaps of Fredman and Tarjan [5]. With this approach, queries are answered in $O(m + n \log n)$ worst-case time and updates require optimal time.

The dynamic maintenance of shortest paths has a long history, and the first papers date back to 1967 [11,12,17]. In 1985 Even and Gazit [4] presented algorithms for maintaining shortest paths on directed graphs with arbitrary real weights. The worst-case bounds of their algorithm for edge deletions were comparable to recomputing APSP from scratch. Also Ramalingam and Reps [15,16] and Frigioni et al. [7,8] considered dynamic shortest path algorithms with real weights, but in a different model. Namely, the running time of their algorithm is analyzed in terms of the output change rather than the input size (*output bounded complexity*). Again, in the worst case the running times of output-bounded dynamic algorithms are comparable to recomputing APSP from scratch.

The first decremental algorithm that was provably faster than recomputing from scratch was devised by King for the special case of graphs with integer edge weights less than C: her algorithm can update shortest paths in a graph subject to a sequence of $\Omega(n^2)$ deletions in $O(C \cdot n^2)$ amortized time per deletion [9]. Later, Demetrescu and Italiano showed how to deal with graphs with real nonnegative edge weights in $O(n^2 \log n)$ amortized time per deletion [2] in a sequence of $\Omega(m/n)$ operations. Both algorithms work in the more general case where edges are not deleted from the graph, but their weight is increased at each update. Moreover, since they update shortest paths explicitly after each deletion, queries are answered in optimal time at any time during a sequence of operations.

Key Results

The decremental APSP algorithm by Demetrescu and Italiano hinges upon the notion of locally shortest paths [2].

Definition 1 A path is *locally shortest* in a graph if all of its proper subpaths are shortest paths.

Notice that by the optimal-substructure property, a shortest path is locally shortest. The main idea of the algorithm is to keep information about locally shortest paths in a graph subject to edge deletions. The following theorem derived from [2] bounds the number of changes in the set of locally shortest paths due to an edge deletion:

Theorem 1 *If shortest paths are unique in the graph, then the number of paths that start or stop being shortest at each deletion is $O(n^2)$ amortized over $\Omega(m/n)$ update operations.*

The result of Theorem 1 is purely combinatorial and assumes that shortest paths are unique in the graph. The latter can be easily achieved using any consistent tie-breaking strategy (see, e. g., [2]). It is possible to design a deletions-only algorithm that pays only $O(\log n)$ time per change in the set of locally shortest paths, using a simple modification of Dijkstra's algorithm [3]. Since by Theorem 1 the amortized number of changes is bounded by $O(n^2)$, this yields the following result:

Theorem 2 *Consider a graph with n vertices and an initial number of m edges subject to a sequence of $\Omega(m/n)$ edge deletions. If shortest paths are unique and edge weights are non-negative, it is possible to support each* `delete` *operation in $O(n^2 \log n)$ amortized time, each* `distance` *query in O(1) worst-case time, and each* `path` *query in $O(\ell)$ worst-case time, where ℓ is the number of vertices in the reported shortest path. The space used is O(mn).*

Applications

Application scenarios of dynamic shortest paths include network optimization [1], document formatting [10], routing in communication systems, robotics, incremental compilation, traffic information systems [18], and dataflow analysis. A comprehensive review of real-world applications of dynamic shortest path problems appears in [14].

URL to Code

An efficient C language implementation of the decremental algorithm addressed in Section "Key Results" is available at the URL: http://www.dis.uniroma1.it/~demetres/experim/dsp.

Cross References

- ▶ All Pairs Shortest Paths in Sparse Graphs
- ▶ All Pairs Shortest Paths via Matrix Multiplication
- ▶ Fully Dynamic All Pairs Shortest Paths

Recommended Reading

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs, NJ (1993)
2. Demetrescu, C., Italiano, G.: A new approach to dynamic all pairs shortest paths. J. Assoc. Comp. Mach. **51**, 968–992 (2004)
3. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
4. Even, S., Gazit, H.: Updating distances in dynamic graphs. Meth. Op. Res. **49**, 371–387 (1985)
5. Fredman, M., Tarjan, R.: Fibonacci heaps and their use in improved network optimization algorithms. J. ACM **34**, 596–615 (1987)
6. Fredman, M.L.: New bounds on the complexity of the shortest path problems. SIAM J. Comp. **5**(1), 87–89 (1976)
7. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica **22**, 250–274 (1998)
8. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic algorithms for maintaining shortest paths trees. J. Algorithm **34**, 351–381 (2000)
9. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99), pp. 81–99. IEEE Computer Society, New York, USA (1999)
10. Knuth, D., Plass, M.: Breaking paragraphs into lines. Software-Practice Exp. **11**, 1119–1184 (1981)
11. Loubal, P.: A network evaluation procedure. Highway Res. Rec. **205**, 96–109 (1967)
12. Murchland, J.: The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph, tech. rep., LBS-TNT-26, London Business School. Transport Network Theory Unit, London, UK (1967)
13. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theor. Comp. Sci. **312**, 47–74 (2003) special issue of selected papers from ICALP (2002)
14. Ramalingam, G.: Bounded incremental computation. Lect. Note Comp. Sci. 1089 (1996)
15. Ramalingam, G., Reps, T.: An incremental algorithm for a generalization of the shortest path problem. J. Algorithm **21**, 267–305 (1996)
16. Ramalingam, G., Reps, T.: On the computational complexity of dynamic graph problems. Theor. Comp. Sci. **158**, 233–277 (1996)
17. Rodionov, V.: The parametric problem of shortest distances. USSR Comp. Math. Math. Phys. **8**, 336–343 (1968)
18. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: an empirical case study from public railroad transport. In: Proc. 3rd Workshop on Algorithm Engineering (WAE'99), pp. 110–123. Notes in Computer Science 1668. London, UK (1999)
19. Takaoka, T.: A new upper bound on the complexity of the all pairs shortest path problem. Inf. Proc. Lett. **43**, 195–199 (1992)

Degree-Bounded Planar Spanner with Low Weight

2005; Song, Li, Wang

WEN-ZHAN SONG[1], XIANG-YANG LI[2], WEIZHAO WANG[3]
[1] School of Engineering and Computer Science, Washington State University, Vancouver, WA, USA
[2] Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA
[3] Google Inc, Irvine, CA, USA

Keywords and Synonyms

Unified energy-efficient unicast and broadcast topology control

Problem Definition

An important requirement of wireless ad hoc networks is that they should be self-organizing, and transmission ranges and data paths may need to be dynamically restructured with changing topology. Energy conservation and network performance are probably the most critical issues in wireless *ad hoc* networks, because wireless devices are usually powered by batteries only and have limited computing capability and memory. Hence, in such a dynamic and resource-limited environment, each wireless node needs to locally select communication neighbors and adjust its transmission power accordingly, such that all nodes together self-form a topology that is energy efficient for both unicast and broadcast communications.

To support energy-efficient unicast, the topology is preferred to have the following features in the literature:

1. POWER SPANNER: [1,9,13,16,17] Formally speaking, a subgraph H is called a *power spanner* of a graph G if there is a positive real constant ρ such that for any two nodes, the power consumption of the shortest path in H is at most ρ times of the power consumption of the shortest path in G. Here ρ is called the *power stretch factor* or *spanning ratio*.
2. DEGREE BOUNDED: [1,9,11,13,16,17] It is also desirable that the logical node degree in the constructed topology is bounded from above by a small constant. Bounded logical degree structures find applications in Bluetooth wireless networks since a *master* node can have only seven active slaves simultaneously. A structure with small logical node degree will save the cost of updating the routing table when nodes are mobile. A structure with a small degree and using shorter links could improve the overall network throughout [6].
3. PLANAR:[1,4,13,14,16] A network topology is also preferred to be planar (no two edges crossing each other in the graph) to enable some localized routing algorithms to work correctly and efficiently, such as *Greedy Face Routing* (GFG) [2], *Greedy Perimeter Stateless Routing* (GPSR) [5], *Adaptive Face Routing* (AFR) [7], and *Greedy Other Adaptive Face Routing* (GOAFR) [8]. Notice that with planar network topology as the underlying routing structure, these localized routing protocols guarantee the message delivery without using a routing table: each intermediate node can decide which logical neighboring node to forward the packet to using only local information and the position of the source and the destination.

To support energy-efficient broadcast [15], the locally constructed topology is preferred to be *low-weighted* [10,12]: the total link length of the final topology is within a constant factor of that of *EMST*. Recently, several localized algorithms [10,12] have been proposed to construct low-weighted structures, which indeed approximate the energy efficiency of EMST as the network density increases. However, none of them is power efficient for unicast routing.

Before this work, all known topology control algorithms could not support power efficient unicast and broadcast in the same structure. It is indeed challenging to design a unified topology, especially due to the trade off between spanner and low weight property. The main contribution of this algorithm is to address this issue.

Key Results

This algorithm is the *first* localized topology control algorithm for all nodes to maintain a *unified* energy-efficient topology for unicast and broadcast in wireless ad hoc/sensor networks. In one single structure, the following network properties are guaranteed:

1. **Power efficient unicast**: given any two nodes, there is a path connecting them in the structure with total power cost no more than $2\rho + 1$ times the power cost of any path connecting them in the original network. Here $\rho > 1$ is some constant that will be specified later in this algorithm. It assumes that each node u can adjust its power sufficiently to cover its next-hop v on any selected path for unicast.
2. **Power efficient broadcast**: the power consumption for broadcast is within a constant factor of the optimum among all *locally* constructed structures. As proved in [10], to prove this, it equals to prove that the structure is *low-weighted*. Here we called a structure low-weigthed, if its total edge length is within a constant factor of the total length of the Euclidean Minimum Spanning

1: First, each node self-constructs the Gabriel graph GG locally. The algorithm to construct GG locally is well-known, and a possible implementation may refer to [13]. Initially, all nodes mark themselves WHITE, i. e., *unprocessed*.
2: Once a WHITE node u has the smallest ID among all its WHITE neighbors in $N(u)$, it uses the following strategy to select neighbors:
   1. Node u first sorts all its BLACK neighbors (if available) in $N(u)$ in the distance-increasing order, then sorts all its WHITE neighbors (if available) in $N(u)$ similarly. The sorted results are then restored to $N(u)$, by first writing the sorted list of BLACK neighbors then appending the sorted list of WHITE neighbors.
   2. Node u scans the sorted list $N(u)$ from left to right. In each step, it keeps the current pointed neighbor w in the list, while deletes every *conflicted* node v in the remainder of the list. Here a node v is conflicted with w means that node v is in the θ-dominating region of node w. Here $\theta = 2\pi/k$ ($k \geq 9$) is an adjustable parameter.

   Node u then marks itself BLACK, i. e. *processed*, and notifies each deleted neighboring node v in $N(u)$ by a broadcasting message UPDATEN.
3: Once a node v receives the message UPDATEN from a neighbor u in $N(v)$, it checks whether itself is in the nodes set for deleting: if so, it deletes the sending node u from list $N(v)$, otherwise, marks u as BLACK in $N(v)$.
4: When all nodes are processed, all selected links $\{uv | v \in N(u), \forall v \in GG\}$ form the final network topology, denoted by $S\Theta GG$. Each node can shrink its transmission range as long as it sufficiently reaches its farthest neighbor in the final topology.

Degree-Bounded Planar Spanner with Low Weight, Algorithm 1
***SΘGG*: Power-Efficient Unicast Topology**

Tree (EMST). For broadcast or generally multicast, it assumes that each node u can adjust its power sufficiently to cover its farthest down-stream node on any selected structure (typically a tree) for multicast.

3. **Bounded logical node degree**: each node has to communicate with at most $k-1$ logical neighbors, where $k \geq 9$ is an adjustable parameter.
4. **Bounded average physical node degree**: the expected average physical node degree is at most a small constant. Here the physical degree of a node u in a structure H is defined as the number of nodes inside the disk centered at u with radius $\max_{uv \in H} \|uv\|$.
5. **Planar**: there are no edges crossing each other. This enables several localized routing algorithms, such as [2,5,7,8], to be performed on top of this structure and guarantee the packet delivery without using the routing table.
6. **Neighbors *Θ*-separated**: the directions between any two logical neighbors of any node are separated by at least an angle θ, which reduces the communication interferences.

It is the *first* known *localized* topology control strategy for all nodes together to maintain such a *single* structure with these desired properties. Previously, only a centralized algorithm was reported in [1]. The first step is Algorithm 1 that can construct a power-efficient topology for unicast, then it extends to the final algorithm (Algorithm 2) that can support power-efficient broadcast at the same time.

Definition 1 (θ-Dominating Region) For each neighbor node v of a node u, the *θ-dominating region* of v is the 2θ-cone emanated from u, with the edge uv as its axis.

Let $N_{\mathrm{UDG}}(u)$ be the set of neighbors of node u in *UDG*, and let $N(u)$ be the set of neighbors of node u in the final topology, which is initialized as the set of neighbor nodes in GG.

Algorithm 1 constructs a degree-$(k-1)$ planar power spanner.

Lemma 1 *Graph $S\Theta GG$ is connected if the underlying graph GG is connected. Furthermore, given any two nodes u and v, there exists a path $\{u, t_1, \ldots, t_r, v\}$ connecting them such that all edges have length less than $\sqrt{2}\|uv\|$.*

Theorem 2 *The structure $S\Theta GG$ has node degree at most $k-1$ and is planar power spanner with neighbors Θ-separated. Its power stretch factor is at most $\rho = \sqrt{2}^{\beta} / (1 - (2\sqrt{2}\sin\frac{\pi}{k})^{\beta})$, where $k \geq 9$ is an adjustable parameter.*

Obviously, the construction is consistent for two endpoints of each edge: if an edge uv is kept by node u, then it is also kept by node v. It is worth mentioning that, the number 3 in criterion $\|xy\| > \max(\|uv\|, 3\|ux\|, 3\|vy\|)$ is carefully selected.

Theorem 3 *The structure $LS\Theta GG$ is a degree-bounded planar spanner. It has a constant power spanning ratio*

1: All nodes together construct the graph $SΘGG$ in a localized manner, as described in Algorithm 1. Then, each node marks its incident edges in $SΘGG$ *unprocessed*.
2: Each node u locally broadcasts its incident edges in $SΘGG$ to its one-hop neighbors and listens to its neighbors. Then, each node x can learn the existence of the set of 2-hop links $E_2(x)$, which is defined as follows: $E_2(x) = \{uv \in SΘGG \mid u \text{ or } v \in N_{\text{UDG}}(x)\}$. In other words, $E_2(x)$ represents the set of edges in $SΘGG$ with at least one endpoint in the transmission range of node x.
3: Once a node x learns that its *unprocessed* incident edge xy has the smallest ID among all *unprocessed* links in $E_2(x)$, it will delete edge xy if there exists an edge $uv \in E_2(x)$ (here both u and v are different from x and y), such that $\|xy\| > \max(\|uv\|, 3\|ux\|, 3\|vy\|)$; otherwise it simply marks edge xy *processed*. Here assume that $uvyx$ is the convex hull of u, v, x and y. Then the link status is broadcasted to all neighbors through a message UPDATESTATUS(XY).
4: Once a node u receives a message UPDATESTATUS(XY), it records the status of link xy at $E_2(u)$.
5: Each node repeats the above two steps until all edges have been *processed*. Let $LSΘGG$ be the final structure formed by all remaining edges in $SΘGG$.

Degree-Bounded Planar Spanner with Low Weight, Algorithm 2
Construct *LSΘGG*: Planar Spanner with Bounded Degree and Low Weight

$2\rho + 1$, *where* ρ *is the power spanning ratio of* $SΘGG$. *The node degree is bounded by* $k - 1$ *where* $k \geq 9$ *is a customizable parameter in* $SΘGG$.

Theorem 4 *The structure LSΘGG is low-weighted.*

Theorem 5 *Assuming that both the ID and the geometry position can be represented by* $\log n$ *bits each, the total number of messages during constructing the structure LSΘGG is in the range of* $[5n, 13n]$, *where each message has at most* $O(\log n)$ *bits.*

Compared with previous known low-weighted structures [10,12], $LSΘGG$ not only achieves more desirable properties, but also costs much less messages during construction. To construct $LSΘGG$, each node only needs to collect the information $E_2(x)$ which costs at most $6n$ messages for n nodes. The Algorithm 2 can be generally applied to any known degree-bounded planar spanner to make it low-weighted while keeping all its previous properties, except increasing the spanning ratio from ρ to $2\rho + 1$ theoretically.

In addition, the expected average node interference in the structure is bounded by a small constant. This is significant on its own due to the following reasons: it has been taken for granted that "*a network topology with small logical node degree will guarantee a small interference*" and recently Burkhart et al. [3] showed that this is not true generally. This work also shows that, although generally a small logical node degree cannot guarantee a small interference, the expected average interference is indeed small if the logical communication neighbors are chosen carefully.

Theorem 6 *For a set of nodes produced by a Poisson point process with density n, the expected maximum node interferences of EMST, GG, RNG, and Yao are at least* $\Theta(\log n)$.

Theorem 7 *For a set of nodes produced by a Poisson point process with density n, the expected average node interferences of EMST are bounded from above by a constant.*

This result also holds for nodes deployed with uniform random distribution.

Applications

Localized topology control in wireless ad hoc networks are critical mechanisms to maintain network connectivity and provide feedback to communication protocols. The major traffic in networks are unicast communications. There is a compelling need to conserve energy and improve network performance by maintaining an energy-efficient topology in localized ways. This algorithm achieves this by choosing relatively smaller power levels and size of communication neighbors for each node (e. g., reducing interference). Also, broadcasting is often necessary in MANET routing protocols. For example, many unicast routing protocols such as Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Zone Routing Protocol (ZRP), and Location Aided Routing (LAR) use broadcasting or a derivation of it to establish routes. It is highly important to use power-efficient broadcast algorithms for such networks since wireless devices are often powered by batteries only.

Cross References

- ▶ Applications of Geometric Spanner Networks
- ▶ Geometric Spanners
- ▶ Planar Geometric Spanners
- ▶ Sparse Graph Spanners

Recommended Reading

1. Bose, P., Gudmundsson, J., Smid, M.: Constructing plane spanners of bounded degree and low weight. In: Proceedings of European Symposium of Algorithms, University of Rome, 17–21 September 2002
2. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. ACM/Kluwer Wireless Networks **7**(6), 609–616 (2001). 3rd int. Workshop on Discrete Algorithms and methods for mobile computing and communications, 48–55 (1999)
3. Burkhart, M., von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Does topology control reduce interference. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Tokyo, 24–26 May 2004
4. Gabriel, K.R., Sokal, R.R.: A new statistical approach to geographic variation analysis. Syst. Zool. **18**, 259–278 (1969)
5. Karp, B., Kung, H.T.: Gpsr: Greedy perimeter stateless routing for wireless networks. In: Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), Boston, 6–11 August 2000
6. Kleinrock, L., Silvester, J.: Optimum transmission radii for packet radio networks or why six is a magic number. In: Proceedings of the IEEE National Telecommunications Conference, pp. 431–435, Birmingham, 4–6 December 1978
7. Kuhn, F., Wattenhofer, R., Zollinger, A.: Asymptotically optimal geometric mobile ad-hoc routing. In: International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), Atlanta, 28 September 2002
8. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-case optimal and average-case efficient geometric ad-hoc routing. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc) Anapolis, 1–3 June 2003
9. Li, L., Halpern, J.Y., Bahl, P., Wang, Y.-M., Wattenhofer, R.: Analysis of a cone-based distributed topology control algorithms for wireless multi-hop networks. In: PODC: ACM Symposium on Principle of Distributed Computing, Newport, 26–29 August 2001
10. Li, X.-Y.: Approximate MST for UDG locally. In: COCOON, Big Sky, 25–28 July 2003
11. Li, X.-Y., Wan, P.-J., Wang, Y., Frieder, O.: Sparse power efficient topology for wireless networks. In: IEEE Hawaii Int. Conf. on System Sciences (HICSS), Big Island, 7–10 January 2002
12. Li, X.-Y., Wang, Y., Song, W.-Z., Wan, P.-J., Frieder, O.: Localized minimum spanning tree and its applications in wireless ad hoc networks. In: IEEE INFOCOM, Hong Kong, 7–11 March 2004
13. Song, W.-Z., Wang, Y., Li, X.-Y. Frieder, O.: Localized algorithms for energy efficient topology in wireless ad hoc networks. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Tokyo, 24–26 May 2004
14. Toussaint, G.T.: The relative neighborhood graph of a finite planar set. Pattern Recognit. **12**(4), 261–268 (1980)
15. Wan, P.-J., Calinescu, G., Li, X.-Y., Frieder, O.: Minimum-energy broadcast routing in static ad hoc wireless networks. ACM Wireless Networks (2002), To appear, Preliminary version appeared in IEEE INFOCOM, Anchorage, 22–26 April 2001
16. Wang, Y., Li, X.-Y.: Efficient construction of bounded degree and planar spanner for wireless networks. In: ACM DIALM-POMC Joint Workshop on Foundations of Mobile Computing, San Diego, 19 September 2003
17. Yao, A.C.-C.: On constructing minimum spanning trees in k-dimensional spaces and related problems. SIAM J. Comput. **11**, 721–736 (1982)

Degree-Bounded Trees

1994; Fürer, Raghavachari

Martin Fürer
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Keywords and Synonyms

Bounded degree spanning trees; Bounded degree Steiner trees

Problem Definition

The problem is to construct a spanning tree of small degree for a connected undirected graph $G = (V, E)$. In the Steiner version of the problem, a set of distinguished vertices $D \subseteq V$ is given along with the input graph G. A Steiner tree is a tree in G which spans at least the set D.

As finding a spanning or Steiner tree of the smallest possible degree Δ^* is *NP*-hard, one is interested in approximating this minimization problem. For many such combinatorial optimization problems, the goal is to find an approximation in polynomial time (a constant or larger factor). For the spanning and Steiner tree problems, the iterative polynomial time approximation algorithms of Fürer and Raghavachari [8] (see also [14]) find much better solutions. The degree Δ of the solution tree is at most $\Delta^* + 1$.

There are very few natural *NP*-hard optimization problems for which the optimum can be achieved up to an additive term of 1. One such problem is coloring a planar graph, where coloring with four colors can be done in polynomial time. On the other hand, 3-coloring is *NP*-complete even for planar graphs. An other such problem is edge coloring a graph of degree Δ. While coloring with $\Delta + 1$ colors is always possible in polynomial time, Δ edge coloring is *NP*-complete.

Chvátal [3] has defined the *toughness* $\tau(G)$ of a graph as the minimum ratio $|X|/c(X)$ such that the subgraph of G induced by $V \backslash X$ has $c(X) \geq 2$ connected compo-

nents. The inequality $1/\tau(G) \leq \Delta^*$ immediately follows. Win [17] has shown that $\Delta^* < \frac{1}{\tau(G)} + 3$; i. e., the inverse of the toughness is actually a good approximation of Δ^*.

A set X, such that the ratio $|X|/c(X)$ is the toughness $\tau(G)$, can be viewed as witnessing the upper bound $|X|/c(X)$ on $\tau(G)$ and therefore the lower bound $c(X)/|X|$ on Δ^*. Strengthening this notion, Fürer and Raghavachari [8] define X to be a witness set for $\Delta^* \geq d$ if d is the smallest integer greater or equal to $(|X| + c(X) - 1)/|X|$. Their algorithm not only outputs a spanning tree, but also a witness set X, proving that its degree is at most $\Delta^* + 1$.

Key Results

The minimum degree spanning tree and Steiner tree problems are easily seen to be *NP*-hard, as they contain the Hamiltonian path problem. Hence, we cannot expect a polynomial time algorithm to find a solution of minimal possible degree Δ^*. The same argument also shows that an approximation by a factor less than 3/2 is impossible in polynomial time unless $P = NP$.

Initial approximation algorithms obtained solutions of degree $O(\Delta^* \log n)$ [6], where $n = |V|$ is the number of vertices. The optimal result for the spanning tree case has been obtained by Fürer and Raghavachari [7, 8].

Theorem 1 *Let Δ^* be the degree of an unknown minimum degree spanning tree of an input graph $G = (V, E)$. There is a polynomial time approximation algorithm for the minimum degree spanning tree problem that finds a spanning tree of degree at most $\Delta^* + 1$.*

Later this result has been extended to the Steiner tree case [8].

Theorem 2 *Assume a Steiner tree problem is defined by a graph $G = (V, E)$ and an arbitrary subset D of vertices V. Let Δ^* be the degree of an unknown minimum degree Steiner tree of G spanning at least the set D. There is a polynomial time approximation algorithm for the minimum degree Steiner tree problem that finds a Steiner tree of degree at most $\Delta^* + 1$.*

Both approximation algorithms run in time $O(mn \cdot \log n\, \alpha(m, n))$, where m is the number of edges and α is the inverse Ackermann function.

Applications

Some possible direct applications are in networks for non-critical broadcasting, where it might be desirable to bound the load per node, and in designing power grids, where the cost of splitting increases with the degree. Another major benefit of a small degree network is limiting the effect of node failure.

Furthermore, the main results on approximating the minimum degree spanning and Steiner tree problems have been the basis for approximating various network design problems, sometimes involving additional parameters.

Klein, Krishnan, Raghavachari and Ravi [11] find 2-connected subgraphs of approximately minimal degree in 2-connected graphs, as well as approximately minimal degree spanning trees (branchings) in directed graphs. Their algorithms run in quasi-polynomial time, and approximate the degree Δ^* by $(1 + \epsilon)\Delta^* + O(\log_{1+\epsilon} n)$.

Often the goal is to find a spanning tree that simultaneously has a small degree and a small weight. For a graph having an minimum weight spanning tree (MST) of degree Δ^* and weight w, Fischer [5] finds a spanning tree with degree $O(\Delta^* + \log n)$ and weight w, (i. e., an MST of small weight) in polynomial time.

Könemann and Ravi [12,13] provide a bi-criteria approximation. For a given $B^* \geq \Delta^*$, let w be the minimum weight of any spanning tree of degree at most B^*. The polynomial time algorithm finds a spanning tree of degree $O(B^* + \log n)$ and weight $O(w)$. In the second paper, the algorithm adapts to the case of a different degree bound on each vertex. Chaudhuri et al. [2] further improved this result to approximate both the degree B^* and the weight w by a constant factor.

In another extension of the minimum degree spanning tree problem, Ravi and Singh [15] have obtained a strict generalization of the $\Delta^* + 1$ spanning tree approximation [8]. Their polynomial time algorithm finds an MST of degree $\Delta^* + k$ for the case of a graph with k distinct weights on the edges.

Recently, there have been some drastic improvements. Again, let w be the minimum cost of a spanning tree of given degree B^*. Goemans [9] obtains a spanning tree of cost w and degree $B^* + 2$. Finally, Singh and Lau [16] decrease the degree to $B^* + 1$ and also handle individual degree bounds Δ_v^* for each vertex v in the same way.

Interesting approximation algorithms are also known for the 2-dimensional Euclidian minimum weight bounded degree spanning tree problem, where the vertices are points in the plane and edge weights are the Euclidian distances. Khuller, Raghavachari, and Young [10] show factor 1.5 and 1.25 approximations for degree bounds 3 and 4 respectively. These bounds have later been improved slightly by Chan [1]. Slightly weaker results are obtained by Fekete et al. [4], using flow-based methods, for the more general case where the weight function just satisfies the triangle inequality.

Open Problems

The time complexity of the minimum degree spanning and Steiner tree algorithms [8] is $O(mn\,\alpha(m,n)\log n)$. Can it be improved to $O(mn)$? In particular, what can be gained by initially selecting a reasonable Steiner tree with some greedy technique instead of starting the iteration with an arbitrary Steiner tree?

Is there an efficient parallel algorithm that can obtain a $\Delta^* + 1$ approximation in poly-logarithmic time? Fürer and Raghavachari [6] have obtained such an NC-algorithm, but only with a factor $O(\log n)$ approximation of the degree.

Cross References

- ▶ Fully Dynamic Connectivity
- ▶ Graph Connectivity
- ▶ Minimum Energy Cost Broadcasting in Wireless Networks
- ▶ Minimum Spanning Trees
- ▶ Steiner Forest
- ▶ Steiner Trees

Recommended Reading

1. Chan, T.M.: Euclidean bounded-degree spanning tree ratios. Discret. Comput. Geom. **32**(2), 177–194 (2004)
2. Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: A push-relabel algorithm for approximating degree bounded MSTs. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006), Part I. LNCS, vol. 4051, pp. 191–201. Springer, Berlin (2006)
3. Chvátal, V.: Tough graphs and Hamiltonian circuits. Discret. Math. **5**, 215–228 (1973)
4. Fekete, S.P., Khuller, S., Klemmstein, M., Raghavachari, B., Young, N.: A network-flow technique for finding low-weight-bounded-degree spanning trees. In: Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference (IPCO 1996) and J. Algorithms **24**(2), 310–324 (1997)
5. Fischer, T.: Optimizing the degree of minimum weight spanning trees, Technical Report TR93–1338. Cornell University, Computer Science Department (1993)
6. Fürer, M., Raghavachari, B.: An NC approximation algorithm for the minimum-degree spanning tree problem. In: Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing, 1990, pp. 174–281
7. Fürer, M., Raghavachari, B.: Approximating the minimum degree spanning tree to within one from the optimal degree. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1992), 1992, pp. 317–324
8. Fürer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. J. Algorithms **17**(3), 409–423 (1994)
9. Goemans, M.X.: Minimum bounded degree spanning trees. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 2006, pp. 273–282
10. Khuller, S., Raghavachari, B., Young, N.: Low-degree spanning trees of small weight. SIAM J. Comput. **25**(2), 355–368 (1996)
11. Klein, P.N., Krishnan, R., Raghavachari, B., Ravi, R.: Approximation algorithms for finding low-degree subgraphs. Networks **44**(3), 203–215 (2004)
12. Könemann, J., Ravi, R.: A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. SIAM J. Comput. **31**(6), 1783–1793 (2002)
13. Könemann, J., Ravi, R.: Primal-dual meets local search: Approximating MSTs with nonuniform degree bounds. SIAM J. Comput. **34**(3), 763–773 (2005)
14. Raghavachari, B.: Algorithms for finding low degree structures. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems. pp. 266–295. PWS Publishing Company, Boston (1995)
15. Ravi, R., Singh, M.: Delegate and conquer: An LP-based approximation algorithm for minimum degree MSTs. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006) Part I. LNCS, vol. 4051, pp. 169–180. Springer, Berlin (2006)
16. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: Proceedings of the thirty-ninth Annual ACM Symposium on Theory of Computing (STOC 2007), New York, NY, 2007, pp. 661–670
17. Win, S.: On a connection between the existence of k-trees and the toughness of a graph. Graphs Comb. **5**(1), 201–205 (1989)

Deterministic Broadcasting in Radio Networks

2000; Chrobak, Gąsieniec, Rytter

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

Keywords and Synonyms

Wireless networks; Dissemination of information; One-to-all communication

Problem Definition

One of the most fundamental communication problems in wired as well as wireless networks is **broadcasting**, where one distinguished source node has a message that needs to be sent to all other nodes in the network.

The radio network abstraction captures the features of distributed communication networks with multi-access channels, with minimal assumptions on the channel model and processors' knowledge. Directed edges model uni-directional links, including situations in which one of two adjacent transmitters is more powerful than the

other. In particular, there is no feedback mechanism (see, for example, [13]). In some applications, collisions may be difficult to distinguish from the noise that is normally present on the channel, justifying the need for protocols that do not depend on the reliability of the collision detection mechanism (see [9,10]). Some network configurations are subject to frequent changes. In other networks, topologies could be unstable or dynamic; for example, when mobile users are present. In such situations, algorithms that do not assume any specific topology are more desirable.

More formally a radio network is a directed graph where by n we denote the number of nodes in this graph. If there is an edge from u to v, then we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. Each node is assigned a unique identifier from the set $\{1, 2, \ldots, n\}$. In the broadcast problem, one node, for example node 1, is distinguished as the *source node*. Initially, the nodes do not possess any other information. In particular, they do not know the network topology.

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

All efficient radio broadcasting algorithms are based on the following purely combinatorial concept of selectors.

Selectors Consider subsets of $\{1, \ldots, n\}$. We say that a set S *hits* a set X iff $|S \cap X| = 1$, and that S *avoids* Y iff $S \cap Y = \emptyset$. A family S of sets is a *w-selector* if it satisfies the following property:

($*$) For any two disjoint sets X, Y with $w/2 \leq |X| \leq w$, $|Y| \leq w$, there is a set in S which hits X and avoids Y.

A complete layered network is a graph consisting of layers $L_0, \ldots, L_{m-1}$, in which each node in layer L_i is directly connected to every node in layer L_{i+1}, for all $i = 0, \ldots, m-1$. The layer L_0 contains only the source node s.

Key Results

Theorem 1 ([5]) *For all positive integers w and n, s.t., $w \leq n$ there exists a w-selector $\tilde{S}$ with $O(w \log n)$ sets.*

Theorem 2 ([5]) *There exists a deterministic $O(n \log^2 n)$-time algorithm for broadcasting in radio networks with arbitrary topology.*

Theorem 3 ([5]) *There exists a deterministic $O(n \log n)$-time algorithm for broadcasting in complete layered radio networks.*

Applications

Prior to this work, Bruschi and Del Pinto showed in [1] that radio broadcasting requires time $\Omega(n \log D)$ in the worst case. In [2], Chlebus et al. presented a broadcasting algorithm with time complexity $O(n^{11/6})$ – the first subquadratic upper bound. This upper bound was later improved to $O(n^{5/3} \log^3 n)$ by De Marco and Pelc [8], and by Chlebus et al. [3] to $O(n^{3/2})$ by application of finite geometries.

Recently, Kowalski and Pelc in [12] proposed a faster $O(n \log n \log D)$—time radio broadcasting algorithm, where D is the eccentricity of the network. Later, Czumaj and Rytter showed in [6] how to reduce this bound to $O(n \log^2 D)$. The results presented in [5], see Theorems 1, 2, and 3, as well as further improvements in [6,12] are existential (non-constructive). The proofs are based on the probabilistic method. A discussion on efficient explicit construction of selectors was initiated by Indyk in [11], and then continued by Chlebus and Kowalski in [4].

More careful analysis and further discussion on selectors in the context of *combinatorial group testing* can be found in [7], where DeBonis et al. proved that the size of selectors is $\Theta(w \log \frac{n}{w})$.

Open Problems

The exact complexity of radio broadcasting remains an open problem, although the gap between the lower and upper bounds $\Omega(n \log D)$ and $O(n \log^2 D)$ is now only a factor of $\log D$. Another promising direction for further studies is improvement of efficient explicit construction of selectors.

Recommended Reading

1. Bruschi, D., Del Pinto, M.: Lower Bounds for the Broadcast Problem in Mobile Radio Networks. Distrib. Comput. **10**(3), 129–135 (1997)
2. Chlebus, B.S., Gąsieniec, L., Gibbons, A.M., Pelc, A., Rytter, W.: Deterministic broadcasting in unknown radio networks. Distrib. Comput. **15**(1), 27–38 (2002)
3. Chlebus, M., Gąsieniec, L., Östlin, A., Robson, J.M.: Deterministic broadcasting in radio networks. In: Proc. 27th International Colloquium on Automata, Languages and Programming.LNCS, vol. 1853, pp. 717–728, Geneva, Switzerland (2000)
4. Chlebus, B.S., Kowalski, D.R.: Almost Optimal Explicit Selectors. In: Proc. 15th International Symposium on Fundamentals of Computation Theory, pp. 270–280, Lübeck, Germany (2005)
5. Chrobak, M., Gąsieniec, L., Rytter, W.: Fast Broadcasting and Gossiping in Radio Networks,. In: Proc. 41st Annual Symposium on Foundations of Computer Science, pp. 575–581, Redondo Beach, USA (2000) Full version in J. Algorithms **43**(2) 177–189 (2002)
6. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. J. Algorithms **60**(2), 115–143 (2006)
7. De Bonis, A., Gąsieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. SIAM J. Comput. **34**(5), 1253–1270 (2005)
8. De Marco, G., Pelc, A.: Faster broadcasting in unknown radio networks. Inf. Process. Lett. **79**(2), 53–56 (2001)
9. Ephremides, A., Hajek, B.: Information theory and communication networks: an unconsummated union. IEEE Trans. Inf. Theor. **44**, 2416–2434 (1998)
10. Gallager, R.: A perspective on multiaccess communications. IEEE Trans. Inf. Theor. **31**, 124–142 (1985)
11. Indyk, P.: Explicit constructions of selectors and related combinatorial structures, with applications. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 697–704, San Francisco, USA (2002)
12. Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad hoc radio networks. Distr. Comput. **18**(1), 43–57 (2005)
13. Massey, J.L., Mathys, P.: The collision channel without feedback. IEEE Trans. Inf. Theor. **31**, 192–204 (1985)

Deterministic Searching on the Line

1988; Baeza-Yates, Culberson, Rawlins

Ricardo Baeza-Yates
Department of Computer Science,
University of Chile,
Santiago, Chile

Keywords and Synonyms

Searching for a point in a line; Searching in one dimension; Searching for a line (or a plane) of known slope in the plane (or a 3D space)

Problem Definition

The problem is to design a strategy for a searcher (or a number of searchers) located initially at some start point on a line to reach an unknown target point. The target point is detected only when a searcher is located on it. There are several variations depending on the information about the target point, how many parallel searchers are available and how they can communicate, and the type of algorithm. The cost of the search algorithm is defined as the distance traveled until finding the point relative to the distance of the starting point to the target. This entry only covers deterministic algorithms.

Key Results

Consider just one searcher. If one knows the direction to the target, the solution is trivial and the relative cost is 1. If one knows the distance to the target, the solution is also simple. Walk that distance to one side and if the target is not found, go back and travel to the other side until the target is found. In the worst case the cost of this algorithm is 3.

If no information is known about the target, the solution is not trivial. The optimal algorithm follows a linear logarithmic spiral with exponent 2 and has cost 9 plus lower order terms. That is, one takes $1, 2, 4, 8, ..., 2^i, ...$ steps to each side in an alternating fashion, each time returning to the origin, until the target is found. This result was first discovered by Gal and rediscovered independently by Baeza-Yates et al.

If one has more searchers, say m, the solution is trivial if they have instantaneous communication. Two searchers walk in opposite directions and the rest stay at the origin. The searcher that finds the target communicates this to all the others. Hence, the cost for all searchers is $m+2$, assuming that all of them must reach the target. If they do not have communication the solution is more complicated and the optimal algorithm is still an open problem.

The searching setting can also be changed, like finding a point in a set of r rays, where the optimal algorithm has cost $1 + 2r^r/(r-1)^{r-1}$, which tends to $1 + 2e \approx 6.44$.

Other variations are possible. For example, if one is interested in the average case one can have a probability distribution for finding the target point, obtaining paradoxical results, as an optimal finite distance algorithm with an infinite number of turning points. On the other hand, in the worst case, if there is a cost d associated with each turn, the optimal distance is 9 OPT + $2d$, where OPT is the distance between the origin and the target. This last case has also been solved for r rays.

The same ideas of doubling in each step can be extended to find a target point in an unknown simple polygon or to find a line with known slope in the plane. The same spiral search can also be used to find an arbitrary line in the plane with cost 13.81. The optimality of this result is still an open problem.

Applications

This problem is a basic element for robot navigation in unknown environments. For example, it arises when a robot needs to find where a wall ends, if the robot can only sense the wall but not see it.

Cross References

► Randomized Searching on Rays or the Line

Recommended Reading

1. Alpern, S., Gal, S.: The Theory of Search Games and Rendevouz. Kluwer Academic Publishers, Dordrecht (2003)
2. Baeza-Yates, R., Culberson, J., Rawlins, G.: Searching in the Plane. Inf. Comput. **106**(2), 234–252 (1993) Preliminary version as Searching with uncertainty. In: Karlsson, R., Lingas, A. (eds.) Proceedings SWAT 88, First Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 318, pp. 176–189. Halmstad, Sweden (1988)
3. Baeza-Yates, R., Schott, R.: Parallel searching in the plane. Comput. Geom. Theor. Appl. **5**, 143–154 (1995)
4. Blum, A., Raghavan, P., Schieber, B.: Navigating in Unfamiliar Geometric Terrain. In: On Line Algorithms, pp. 151–155, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence RI (1992) Preliminary Version in STOC 1991, pp. 494–504
5. Demaine, E., Fekete, S., Gal, S.: Online searching with turn cost. Theor. Comput. Sci. **361**, 342–355 (2006)
6. Gal, S.: Minimax solutions for linear search problems. SIAM J. Appl. Math. **27**, 17–30 (1974)
7. Gal, S.: Search Games, pp. 109–115, 137–151, 189–195. Academic Press, New York (1980)
8. Hipke, C., Icking, C., Klein, R., Langetepe, E.: How to Find a point on a line within a Fixed distance. Discret. Appl. Math. **93**, 67–73 (1999)
9. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. Inf. Comput. **131**(1), 63–79 (1996) Preliminary version in SODA '93, pp. 441–447
10. Lopez-Ortiz, A.: On-Line Target Searching in Bounded and Unbounded Domains: Ph. D. Thesis, Technical Report CS-96-25, Dept. of Computer Sci., Univ. of Waterloo (1996)
11. Lopez-Ortiz, A., Schuierer, S.: The Ultimate Strategy to Search on m Rays? Theor. Comput. Sci. **261**(2), 267–295 (2001)
12. Papadimitriou, C.H., Yannakakis, M.: Shortest Paths without a Map. Theor. Comput. Sci. **84**, 127–150 (1991) Preliminary version in ICALP '89
13. Schuierer, S.: Lower bounds in on-line geometric searching. Comput. Geom. **18**, 37–53 (2001)

Detour

► Dilation of Geometric Networks
► Geometric Dilation of Geometric Networks
► Planar Geometric Spanners

Dictionary-Based Data Compression

1977; Ziv, Lempel

TRAVIS GAGIE, GIOVANNI MANZINI
Department of Computer Science,
University of Eastern Piedmont, Alessandria, Italy

Keywords and Synonyms

LZ compression; Ziv–Lempel compression; Parsing-based compression

Problem Definition

The problem of lossless data compression is the problem of compactly representing data in a format that admits the faithful recovery of the original information. Lossless data compression is achieved by taking advantage of the redundancy which is often present in the data generated by either humans or machines.

Dictionary-based data compression has been "the solution" to the problem of lossless data compression for nearly 15 years. This technique originated in two theoretical papers of Ziv and Lempel [15,16] and gained popularity in the "80s" with the introduction of the Unix tool `compress` (1986) and of the gif image format (1987). Although today there are alternative solutions to the problem of lossless data compression (e. g., Burrows-Wheeler compression and Prediction by Partial Matching), dictionary-based compression is still widely used in everyday applications: consider for example the zip utility and its variants, the modem compression standards V.42bis and V.44, and the transparent compression of pdf documents. The main reason for the success of dictionary-based compression is its unique combination of compression power and compression/decompression speed. The reader should refer to [13] for a review of several dictionary-based compression algorithms and of their main features.

Key Results

Let T be a string drawn from an alphabet Σ. Dictionary-based compression algorithms work by parsing the input into a sequence of substrings (also called words) $T_1, T_2, \ldots, T_d$ and by encoding a compact representation of these substrings. The parsing is usually done incrementally and on-line with the following iterative procedure.

Assume the encoder has already parsed the substrings $T_1, T_2, \ldots, T_{i-1}$. To proceed, the encoder maintains a dictionary of potential candidates for the next word T_i and associates a unique codeword with each of them. Then, it looks at the incoming data, selects one of the candidates, and emits the corresponding codeword. Different algorithms use different strategies for establishing which words are in the dictionary and for choosing the next word T_i. A larger dictionary implies a greater flexibility for the choice of the next word, but also longer codewords. Note that for efficiency reasons the dictionary is usually not built explicitly: the whole process is carried out implicitly using appropriate data structures.

Dictionary-based algorithms are usually classified into two families whose respective ancestors are two parsing strategies, both proposed by Ziv and Lempel and today universally known as LZ78 [16] and LZ77 [15].

The LZ78 Algorithm

Assume the encoder has already parsed the words $T_1, T_2, \ldots, T_{i-1}$, that is, $T = T_1 T_2 \cdots T_{i-1} \hat{T}_i$ for some text suffix $\hat{T}_i$. The LZ78 dictionary is defined as the set of strings obtained by adding a single character to one of the words $T_1, \ldots, T_{i-1}$ or to the empty word. The next word T_i is defined as the longest prefix of $\hat{T}_i$ which is a dictionary word. For example, for $T = aabbaaabaabaabba$ the LZ78 parsing is: $a, ab, b, aa, aba, abaa, bb, a$. It is easy to see that all words in the parsing are distinct, with the possible exception of the last one (in the example the word a). Let T_0 denote the empty word. If $T_i = T_j\alpha$, with $0 \le j < i$ and $\alpha \in \Sigma$, the codeword emitted by LZ78 for T_i will be the pair (j, α). Thus, if LZ78 parses the string T into t words, its output will be bounded by $t \log t + t \log |\Sigma| + \Theta(t)$ bits.

The LZ77 Algorithm

Assume the encoder has already parsed the words $T_1, T_2, \ldots, T_{i-1}$, that is, $T = T_1 T_2 \cdots T_{i-1} \hat{T}_i$ for some text suffix $\hat{T}_i$. The LZ77 dictionary is defined as the set of strings of the form $w\alpha$ where $\alpha \in \Sigma$ and w is a substring of T starting in the already parsed portion of T. The next word T_i is defined as the longest prefix of $\hat{T}_i$ which is a dictionary word. For example, for $T = aabbaaabaabaabba$ the LZ77 parsing is: $a, ab, ba, aaba, abaabb, a$. Note that, in some sense, $T_5 = abaabb$ is defined in terms of itself: it is a copy of the dictionary word $w\alpha$ with w starting at the second a of T_4 and extending into T_5! It is easy to see that all words in the parsing are distinct, with the possible exception of the last one (in the example the word a), and that the number of words in the LZ77 parsing is smaller than in the LZ78 parsing. If $T_i = w\alpha$ with $\alpha \in \Sigma$, the codeword for T_i is the triplet (s_i, ℓ_i, α) where s_i is the distance from the start of T_i to the last occurrence of w in $T_1 T_2 \cdots T_{i-1}$, and $\ell_i = |w|$.

Entropy Bounds

The performance of dictionary-based compressors has been extensively investigated since their introduction. In [15] it is shown that LZ77 is optimal for a certain family of sources, and in [16] it is shown that LZ78 achieves asymptotically the best compression ratio attainable by a finite-state compressor. This implies that, when the input string is generated by an ergodic source, the compression ratio achieved by LZ78 approaches the entropy of the source. More recent work has established similar results for other Ziv–Lempel compressors and has investigated the rate of convergence of the compression ratio to the entropy of the source (see [14] and references therein).

It is possible to prove compression bounds without probabilistic assumptions on the input, using the notion of *empirical entropy*. For any string T, the order k empirical entropy $H_k(T)$ is the maximum compression one can achieve using a uniquely decodable code in which the codeword for each character may depend on the k characters immediately preceding it [6]. The following lemma is a useful tool for establishing upper bounds on the compression ratio of dictionary-based algorithms which hold pointwise on every string T.

Lemma 1 ([6, Lemma 2.3]) *Let $T = T_1 T_2 \cdots T_d$ be a parsing of T such that each word T_i appears at most M times. Then, for any $k \ge 0$*

$$d \log d \le |T| H_k(T) + d \log(|T|/d) + d \log M + \Theta(kd + d),$$

where $H_k(T)$ is the k-th order empirical entropy of T. □

Consider, for example, the algorithm LZ78. It parses the input T into t distinct words (ignoring the last word in the parsing) and produces an output bounded by $t \log t + t \log |\Sigma| + \Theta(t)$ bits. Using Lemma 1 and the fact that $t = O(|T|/\log T)$, one can prove that LZ78′s output is at most $|T| H_k(T) + o(|T|)$ bits. Note that the bound holds for any $k \ge 0$: this means that LZ78 is essentially "as powerful" as any compressor that encodes the next character on the basis of a finite context.

Algorithmic Issues

One of the reasons for the popularity of dictionary-based compressors is that they admit linear-time, space-efficient implementations. These implementations sometimes require non-trivial data structures: the reader is referred

to [12] and references therein for further reading on this topic.

Greedy vs. Non-Greedy Parsing

Both LZ78 and LZ77 use a greedy parsing strategy in the sense that, at each step, they select the longest prefix of the unparsed portion which is in the dictionary. It is easy to see that for LZ77 the greedy strategy yields an optimal parsing; that is, a parsing with the minimum number of words. Conversely, greedy parsing is not optimal for LZ78: for any sufficiently large integer m there exists a string that can be parsed to $O(m)$ words and that the greedy strategy parses in $\Omega(m^{3/2})$ words. In [9] the authors describe an efficient algorithm for computing an optimal parsing for the LZ78 dictionary and, indeed, for any dictionary with the prefix-completeness property (a dictionary is prefix-complete if any prefix of a dictionary word is also in the dictionary). Interestingly, the algorithm in [9] is a one-step lookahead greedy algorithm: rather than choosing the longest possible prefix of the unparsed portion of the text, it chooses the prefix that results in the longest advancement in the *next* iteration.

Applications

The natural application field of dictionary-based compressors is lossless data compression (see, for example [13]). However, because of their deep mathematical properties, the Ziv–Lempel parsing rules have also found applications in other algorithmic domains.

Prefetching

Krishnan and Vitter [7] considered the problem of prefetching pages from disk into memory to anticipate users' requests. They combined LZ78 with a pre-existing prefetcher P_1 that is asymptotically at least as good as the best memoryless prefetcher, to obtain a new algorithm P that is asymptotically at least as good as the best finite-state prefetcher. LZ78's dictionary can be viewed as a trie: parsing a string means starting at the root, descending one level for each character in the parsed string and, finally, adding a new leaf. Algorithm P runs LZ78 on the string of page requests as it receives them, and keeps a copy of the simple prefetcher P_1 for each node in the trie; at each step, P prefetches the page requested by the copy of P_1 associated with the node LZ78 is currently visiting.

String Alignment

Crochemore, Landau and Ziv-Ukelson [4] applied LZ78 to the problem of sequence alignment, i. e., finding the cheapest sequence of character insertions, deletions and substitutions that transforms one string T into another T' (the cost of an operation may depend on the character or characters involved). Assume, for simplicity, that $|T| = |T'| = n$. In 1980 Masek and Paterson proposed an $O(n^2/\log n)$-time algorithm with the restriction that the costs be rational; Crochemore et al.'s algorithm allows real-valued costs, has the same asymptotic cost in the worst case, and is asymptotically faster for compressible texts.

The idea behind both algorithms is to break into blocks the matrix $A[1 \ldots n, 1 \ldots n]$ used by the obvious $O(n^2)$-time dynamic programming algorithm. Masek and Paterson break it into uniform-sized blocks, whereas Crochemore et al. break it according to the LZ78 parsing of T and T'. The rationale is that, by the nature of LZ78 parsing, whenever they come to solve a block $A[i \ldots i', j \ldots j']$, they can solve it in $O(i' - i + j' - j)$ time because they have already solved blocks identical to $A[i \ldots i' - 1, j \ldots j']$ and $A[i \ldots i', j \ldots j' - 1]$ [8]. Lifshits, Mozes, Weimann and Ziv-Ukelson [8 recently used a similar approach to speed up the decoding and training of hidden Markov models.

Compressed Full-Text Indexing

Given a text T, the problem of compressed full-text indexing is defined as the task of building an index for T that takes space proportional to the entropy of T and that supports the efficient retrieval of the occurrences of any pattern P in T. In [10] Navarro proposed a compressed full-text index based on the LZ78 dictionary. The basic idea is to keep two copies of the dictionary as tries: one storing the dictionary words, the other storing their reversal. The rationale behind this scheme is the following. Since any non-empty prefix of a dictionary word is also in the dictionary, if the sought pattern P occurs within a dictionary word, then P is a suffix of some word and easy to find in the second dictionary. If P overlaps two words, then some prefix of P is a suffix of the first word—and easy to find in the second dictionary—and the remainder of P is a prefix of the second word—and easy to find in the first dictionary. The case when P overlaps three or more words is a generalization of the case with two words. Recently, Arroyuelo et al. [1] improved the original data structure in [10]. For any text T, the improved index uses $(2 + \epsilon)|T|H_k(T) + o(|T| \log |\Sigma|)$ bits of space, where $H_k(T)$ is the k-th order empirical entropy of T, and reports all occ occurrences of P in T in $O(|P|^2 \log |P| + (|P| + occ) \log |T|)$ time.

Independently of [10], in [5] the LZ78 parsing was used together with the Burrows-Wheeler compression algorithm to design the first full-text index that uses $o(|T|\log|T|)$ bits of space and reports the *occ* occurrences of P in T in $O(|P| + occ)$ time. If $T = T_1T_2\cdots T_d$ is the LZ78 parsing of T, in [5] the authors consider the string $T_\$ = T_1\$T_2\$\cdots\$T_d\$$ where \$ is a new character not belonging to Σ. The string $T_\$$ is then compressed using the Burrows-Wheeler transform. The \$'s play the role of anchor points: their positions in $T_\$$ are stored explicitly so that, to determine the position in T of any occurrence of P, it suffices to determine the position with respect to any of the \$'s. The properties of the LZ78 parsing ensure that the overhead of introducing the \$'s is small, but at the same time the way they are distributed within $T_\$$ guarantees the efficient location of the pattern occurrences.

Related to the problem of compressed full-text indexing is the compressed matching problem in which text and pattern are given together (so the former cannot be preprocessed). Here the task consists in performing string matching in a compressed text without decompressing it. For dictionary-based compressors this problem was first raised in 1994 by A. Amir, G. Benson, and M. Farach, and has received considerable attention since then. The reader is referred to [11] for a recent review of the many theoretical and practical results obtained on this topic.

Substring Compression Problems

Substring compression problems involve preprocessing T to be able to efficiently answer queries about compressing substrings: e. g., how compressible is a given substring s in T? what is s's compressed representation? or, what is the least compressible substring of a given length ℓ? These are important problems in bioinformatics because the compressibility of a DNA sequence may give hints as to its function, and because some clustering algorithms use compressibility to measure similarity. The solutions to these problems are often trivial for simple compressors, such as Huffman coding or run-length encoding, but they are open for more powerful algorithms, such as dictionary-based compressors, BWT compressors, and PPM compressors. Recently, Cormode and Muthukrishnan [3] gave some preliminary solutions for LZ77. For any string s, let $C(s)$ denote the number of words in the LZ77-parsing of s, and let LZ77(s) denote the LZ77-compressed representation of s. In [3] the authors show that, with $O(|T|\,\text{polylog}(|T|))$ time preprocessing, for any substring s of T they can: *a*) compute LZ77(s) in $O(C(s)\log|T|\log\log|T|)$ time, *b*) compute an approximation of $C(s)$ within a factor $O(\log|T|\log^*|T|)$ in $O(1)$ time, *c*) find a substring of length ℓ that is close to being the least compressible in $O(|T|\ell/\log\ell)$ time. These bounds also apply to general versions of these problems, in which queries specify another substring t in T as context and ask about compressing substrings when LZ77 starts with a dictionary already containing the words in the LZ77 parsing of t.

Grammar Generation

Charikar et al. [2] considered LZ78 as an approximation algorithm for the NP-hard problem of finding the smallest context-free grammar that generates only the string T. The LZ78 parsing of T can be viewed as a context-free grammar in which for each dictionary word $T_i = T_j\alpha$ there is a production $X_i \to X_j\alpha$. For example, for T = *aabbaaabaabaabba* the LZ78 parsing is: *a*, *ab*, *b*, *aa*, *aba*, *abaa*, *bb*, *a*, and the corresponding grammar is: $S \to X_1\ldots X_7X_1, X_1 \to a, X_2 \to X_1b, X_3 \to b, X_4 \to X_1a, X_5 \to X_2a, X_6 \to X_5a, X_7 \to X_3b$. Charikar et al. showed LZ78's approximation ratio is in $O((|T|/\log|T|)^{2/3}) \cap \Omega(|T|^{2/3}\log|T|)$; i. e., the grammar it produces has size at most $f(|T|)\cdot m^*$, where $f(|T|)$ is a function in this intersection and m^* is the size of the smallest grammar. They also showed m^* is at least the number of words output by LZ77 on T, and used LZ77 as the basis of a new algorithm with approximation ratio $O(\log(|T|/m^*))$.

URL to Code

The source code of the gzip tool (based on LZ77) is available at the page http://www.gzip.org/. An LZ77-based compression library zlib is available from http://www.zlib.net/. A more recent, and more efficient, dictionary-based compressor is LZMA (Lempel–Ziv Markov chain Algorithm), whose source code is available from http://www.7-zip.org/sdk.html.

Cross References

- ▶ Arithmetic Coding for Data Compression
- ▶ Boosting Textual Compression
- ▶ Burrows–Wheeler Transform
- ▶ Compressed Text Indexing

Recommended Reading

1. Arroyuelo, D., Navarro, G., Sadakane, K.: Reducing the space requirement of LZ-index. In: Proc. 17th Combinatorial Pattern Matching conference (CPM), LNCS no. 4009, pp. 318–329, Springer (2006)
2. Charikar, M., Lehman, E., Liu, D., Panigraphy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Trans. Inf. Theor. **51**, 2554–2576 (2005)

3. Cormode, G., Muthukrishnan, S.: Substring compression problems. In: Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA '05), pp. 321–330 (2005)
4. Crochemore, M., Landau, G., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J. Comput. **32**, 1654–1673 (2003)
5. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM **52**, 552–581 (2005)
6. Kosaraju, R., Manzini, G.: Compression of low entropy strings with Lempel–Ziv algorithms. SIAM J. Comput. **29**, 893–911 (1999)
7. Krishnan, P., Vitter, J.: Optimal prediction for prefetching in the worst case. SIAM J. Comput. **27**, 1617–1636 (1998)
8. Lifshits, Y., Mozes, S., Weimann, O., Ziv-Ukelson, M.: Speeding up HMM decoding and training by exploiting sequence repetitions. Algorithmica to appear doi:10.1007/s00453-007-9128-0
9. Matias, Y., Şahinalp, C.: On the optimality of parsing in dynamic dictionary based data compression. In: Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99), pp. 943–944 (1999)
10. Navarro, G.: Indexing text using the Ziv–Lempel trie. J. Discret. Algorithms **2**, 87–114 (2004)
11. Navarro, G., Tarhio, J.: LZgrep: A Boyer-Moore string matching tool for Ziv–Lempel compressed text. Softw. Pract. Exp. **35**, 1107–1130 (2005)
12. Şahinalp, C., Rajpoot, N.: Dictionary-based data compression: An algorithmic perspective. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 153–167. Academic Press, USA (2003)
13. Salomon, D.: Data Compression: the Complete Reference, 4th edn. Springer, London (2007)
14. Savari, S.: Redundancy of the Lempel–Ziv incremental parsing rule. IEEE Trans. Inf. Theor. **43**, 9–21 (1997)
15. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theor. **23**, 337–343 (1977)
16. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. IEEE Trans. Inf. Theor. **24**, 530–536 (1978)

Dictionary Matching and Indexing (Exact and with Errors)

2004; Cole, Gottlieb, Lewenstein

Moshe Lewenstein
Department of Computer Science, Bar Ilan University, Ramat-Gan, Israel

Keywords and Synonyms

Approximate dictionary matching; Approximate text indexing

Problem Definition

Indexing and *dictionary matching* are generalized models of pattern matching. These models have attained importance with the explosive growth of multimedia, digital libraries, and the Internet.

1. Text Indexing: In text indexing one desires to preprocess a text t, of length n, and to answer where subsequent queries p, of length m, appear in the text t.
2. Dictionary Matching: In dictionary matching one is given a dictionary D of strings $p_1, \ldots, p_d$ to be preprocessed. Subsequent queries provide a query string t, of length n, and ask for each location in t at which patterns of the dictionary appear.

Key Results

Text Indexing

The *indexing* problem assumes a large text that is to be preprocessed in a way that will allow the following efficient future queries. Given a query pattern, one wants to find all text locations that match the pattern in time proportional to the *pattern length and to the number of occurrences*.

To solve the indexing problem, Weiner [14] invented the *suffix tree* data structure (originally called a *position tree*), which can be constructed in linear time, and subsequent queries of length m are answered in time $O(m \log |\Sigma| + tocc)$, where $tocc$ is the number of pattern occurrences in the text.

Weiner's suffix tree in effect solved the indexing problem for exact matching of fixed texts. The construction was simplified by the algorithms of McCreight and, later, Chen and Seiferas. Ukkonen presented an online construction of the suffix tree. Farach presented a linear time construction for large alphabets (specifically, when the alphabet is $\{1, \ldots, n^c\}$, where n is the text size and c is some fixed constant). All results, besides the latter, work by handling one suffix at a time. The latter algorithm uses a divide and conquer approach, dividing the suffixes to be sorted to even-position suffixes and odd-position suffixes. See the entry on Suffix Tree Construction for full details. The standard query time for finding a pattern p in a suffix tree is $O(m \log |\Sigma|)$. By slightly adjusting the suffix tree one can obtain a query time of $O(m + \log n)$, see [12].

Another popular data structure for indexing is suffix arrays. Suffix arrays were introduced by Manber and Myers. Others proposed linear time constructions for linearly bounded alphabets. All three extend the divide and conquer approach presented by Farach. The construction in [11] is especially elegant and significantly simplifies the divide and conquer approach, by dividing the suffix set into three groups instead of two. See the entry on Suffix Array Construction for full details. The query time for suffix arrays is $O(m + \log n)$ achievable by embedding additional lcp (longest common prefix) information into the data structure. See [11] for reference to other solutions. *Suffix Trays* were introduced in [5] as a merge between suf-

fix trees and suffix arrays. The construction time of suffix trays is the same as for suffix trees and suffix arrays. The query time is $O(m + \log|\Sigma|)$.

Solutions for the indexing problem in *dynamic* texts, where insertions and deletions (of single characters or entire substrings) are allowed, appear in several papers, see [2] and references therein.

Dictionary Matching

Dictionary matching is, in some sense, the "inverse" of text indexing. The large body to be preprocessed is a set of patterns, called the *dictionary*. The queries are texts whose length is typically significantly smaller than the dictionary size. It is desired to find all (exact) occurrences of dictionary patterns in the text in time proportional to the *text length and to the number of occurrences*.

Aho and Corasick [1] suggested an automaton-based algorithm that preprocesses the dictionary in time $O(d)$ and answers a query in time $O(n + docc)$, where *docc* is the number of occurrences of patterns within the text. Another approach to solving this problem is to use a generalized suffix tree. A *generalized suffix tree* is a suffix tree for a collection of strings. Dictionary matching is done for the dictionary of patterns. Specifically, a suffix tree is created for the generalized string $p_1\$_1p_2\$_2\cdots\$_dp_d\$_d$, where the $\$_i$'s are not in the alphabet. A randomized solution using a fingerprint scheme was proposed in [3]. In [7] a parallel work-optimal algorithm for dictionary matching was presented. Ferragina and Luccio [8] considered the problem in the external memory model and suggested a solution based upon the String B-tree data structure along with the notion of a certificate for dictionary matching. Two Dimensional Dictionary Matching is another fascinating topic which appears as a separate entry. See also the entry on Multidimensional String Matching.

Dynamic Dictionary Matching: Here one allows insertion and deletion of patterns from the dictionary D. The first solution to the problem was a suffix tree-based method for solving the dynamic dictionary matching problem. Idury and Schäffer [10] showed that the failure function (function mapping from one longest matching prefix to the next longest matching prefix, see [1]) approach and basic scanning loop of the Aho–Corasick algorithm can be adapted to dynamic dictionary matching for improved initial dictionary preprocessing time. They also showed that faster search time can be achieved at the expense of slower dictionary update time.

A further improvement was later achieved by reducing the problem to maintaining a sequence of well-balanced parentheses under certain operations. In [13] an optimal method was achieved based on a labeling paradigm, where labels are given to, sometimes overlapping, substrings of different lengths. The running times are: $O(|D|)$ preprocessing time, $O(m)$ update time, and $O(n + docc)$ time for search. See [13] for other references.

Text Indexing and Dictionary Matching with Errors

In most real-life systems there is a need to allow errors. With the maturity of the solutions for *exact* indexing and *exact* dictionary matching, the quest for *approximate* solutions began. Two of the classical measures for approximating closeness of strings, Hamming distance and Edit distance, were the first natural measures to be considered.

Approximate Text Indexing: For approximate text indexing, given a distance k, one preprocesses a specified text t. The goal is to find all locations ℓ of t within distance k of the query p, i. e. for the Hamming distance all locations ℓ such that the length m substring of t beginning at that location can be made equal to p with at most k character substitutions. (An analogous statement applies for the edit distance.) For $k = 1$ [4] one can preprocess in time $O(n\log^2 n)$ and answer subsequent queries p in time $O(m\sqrt{\log n}\log\log n + occ)$. For small $k \geq 2$, the following naive solutions can be achieved. The first possible solution is to traverse a suffix tree checking all possible configurations of k, or less, mismatches in the pattern. However, while the preprocessing needed to build a suffix tree is cheap, the search is expensive, namely, $O(m^{k+1}|\Sigma|^k + occ)$. Another possible solution, for the Hamming distance measure only, leads to data structures of size approximately $O(n^{k+1})$ embedding all mismatch possibilities into the tree. This can be slightly improved by using the data structures for $k = 1$, which reduce the size to approximately $O(n^k)$.

Approximate Dictionary Matching: The goal is to preprocess the dictionary along with a threshold parameter k in order to support the following subsequent queries: Given a query text, seek all pairs of patterns (from the dictionary) and text locations which match within distance k. Here once again there are several algorithms for the case where $k = 1$ [4,9]. The best solution for this problem has query time $O(m\log\log n + occ)$; the data structure uses space $O(n\log n)$ and can be built in time $O(n\log n)$.

The solutions for $k = 1$ in both problems (Approximate Text Indexing and Approximate Dictionary Matching) are based on the following, elegant idea, presented in Indexing terminology. Say a pattern p matches a text t at location i with one error at location j of p (and at location $i + j - 1$ of t). Obviously, the $j - 1$-length prefix of p matches the aligned substring of t and so does the

$m - j - 1$ length suffix. If t and p are reversed then the $j - 1$-th length prefix of p becomes a $j - 1$-th length suffix of p^R (that is p reverse). Notice that there is a match with, at most one error, if (1) the suffix of p starting at location $j + 1$ matches the (prefix of the) suffix of t starting at location $i + j$ and (2) the suffix of p^R starting at location $m - j + 1$ (the reverse of the $j - 1$-th length prefix of p) matches the (prefix of the) suffix of t^R starting at location $m - i - j + 3$. So, the problem now becomes a search for locations j which satisfy the above. To do so, the above-mentioned solutions, naturally, use two suffix trees, one for the text and one for its reverse (with additional data structure tricks to answer the query fast). In dictionary matching the suffix trees are defined on the dictionary. The problem is that this solution does not carry over for $k \geq 2$. See the introduction of [6] for a full list of references.

Text Indexing and Dictionary Matching within (Small) Distance *k*

Cole et al. [6] proposed a new method that yields a unified solution for approximate text indexing, approximate dictionary matching, and other related problems. However, since the solution is somewhat involved it will be simpler to explain the ideas on the following problem. The desire is to index a text t to allow fast searching for all occurrences of a pattern containing, at most, k don't cares (don't cares are special characters which match all characters).

Once again, there are two possible, relatively straightforward, solutions to be elaborated. The first is to use a suffix tree, which is cheap to preprocess, but causes the search to be expensive, namely, $O(m|\Sigma|^k + occ)$ (if considering k mismatches this would increase to $O(m^{k+1}|\Sigma|^k + occ)$. To be more specific, imagine traversing a path in a suffix tree. Consider the point where a don't care is reached. If in the middle of an edge the only text suffixes (representing substrings) that can match the pattern with this don't care must also go through this edge. So simply continue traversing. However, if at a node, then all the paths leaving this node must be explored. This explains the mentioned time bound.

The second solution is to create a tree that contains all strings that are at Hamming distance k from a suffix. This allows fast search but leads to trees of size exponential in k, namely, $O(n^{k+1})$ size trees. To elaborate, the tree, called a *k-error-trie*, is constructed as follows. First, consider the case for one don't care, i. e. a *1-error-trie*, and then extend it. At any node v a don't care may need to be evaluated. Therefore, create a special subtree branching off this node that represents a don't care at this node. To understand this subtree, note that the subtree (of the suffix tree) rooted at v is actually a compressed trie of (some of the) suffixes of the text. Denote the collection of suffixes S_v. The first character of all these suffixes have to be removed (or, perhaps better imagined as a replacement with a don't care character). Each will be a new suffix of the text. Denote the new collection as S'_v. Now, create a new compressed trie of suffixes for S'_v, calling this new subtree an *error tree*. Do so for every v. The suffix tree along with its error trees is a *1-error-trie*. Turning to queries in the *1-error-trie*, when traversing the *1-error-trie*, do so with the suffix tree up till the don't care at node v. Move into the error tree at node v and continue the traversal of the pattern.

To create a *2-error-trie*, simply take each error tree and construct an error tree for each node within. A *(k+1)-error trie* is created recursively from a *k-error trie*. Clearly the *1-error trie* is of size $O(n^2)$, since any node u in the original suffix tree will appear in all the new subtrees of the *1-error trie* created for each of the nodes v which are ancestors of u. Likewise, the *k-error-trie* is of size $O(n^{k+1})$.

The method introduced in Cole et al. [6] uses the idea of the error trees to form a new data structure, which is called a *k-errata trie*. The *k-errata trie* will be much smaller than $O(n^{k+1})$. However, it comes at the cost of a somewhat slower search time. To understand the *k-errata tries* it is useful to first consider the *1-errata-tries* and to extend. The *1-errata-trie* is constructed as follows. The suffix tree is first decomposed with a centroid path decomposition (which is a decomposition of the nodes into paths, where all nodes along a path have their subtree sizes within a range 2^r and 2^{r+1}, for some integer r). Then, as before, error trees are created for each node v of the suffix tree with the following difference. Namely, consider the subtree, T_v, at node v and consider the edge (v, x) going from v to child x on the centroid path. T_v can be partitioned into two subtrees, $T_x \cup (v, x)$, and T'_v all the rest of T_v. An error tree is created for the suffixes in T'_v. The *1-errata-trie* is the suffix tree with all of its error trees. Likewise, a *(k+1)-errata trie* is created recursively from a *k-errata trie*. The contents of a *k-errata trie* should be viewed as a collection of error trees, k levels deep, where error trees at each level are constructed on the error trees of the previous level (at level 0 there is the original suffix tree). The following lemma helps in obtaining a bound on the size of the *k-errata trie*.

Lemma 1 *Let C be a centroid decomposition of a tree T. Let u be an arbitrary node of T and π be the path from the root to u. There are at most* $\log n$ *nodes v on π for which v and v's parent on π are on different centroid paths.*

The implication is that every node u in the original suffix tree will only appear in $\log n$ error trees of the *1-errata trie* because each ancestor v of u is on the path π from the root to u and only $\log n$ such nodes are on different centroid paths than their children (on π). Hence, u appears in only $\log^k n$ error trees in the *k-errata trie*. Therefore, the size of the *k-errata trie* is $O(n \log^k n)$. Creating the *k-errata tries* in $O(n \log^{k+1} n)$ can be done. To answer queries on a *k-errata trie*, given the pattern with (at most) k don't cares, the 0th level of the *k-errata trie*, i. e. the suffix tree, needs to be traversed. This is to be done until the first don't care, at location j, in the pattern is reached. If at node v in the 0th level of the *k-errata trie*, enter the (1st level) error tree hanging off of v and traverse this error tree from location $j + 2$ of the pattern (until the next don't care is met). However, the error tree hanging off of node v does not contain the subtree hanging off of v that is along the centroid path. Hence, continue traversing the pattern in the 0th level of the *k-errata trie*, starting along the edge on the centroid path leaving v (until the next don't care is met). The search is done recursively for k don't cares and, hence, yields an $O(2^k m)$ time search.

Recall that a solution for indexing text that supports queries of a pattern with k don't cares has been described. Unfortunately, when indexing to support k mismatch queries, not to mention k edit operation queries, the traversal down a *k-errata trie* can be very time consuming as frequent branching is required since an error may occur at any location of the pattern. To circumvent this problem search many error trees in parallel. In order to do so, the error trees have to be grouped together. This needs to be done carefully, see [6] for the full details. Moreover, edit distance needs even more careful handling. The time and space of the algorithms achieved in [6] are as follows:

Approximate Text Indexing: The data structure for mismatches uses space $O(n \log^k n)$, takes time $O(n \log^{k+1} n)$ to build, and answers queries in time $O((\log^k n) \log \log n + m + occ)$. For edit distance, the query time becomes $O((\log^k n) \log \log n + m + 3^k \cdot occ)$. It must be pointed out that this result is mostly effective for constant k.

Approximate Dictionary Matching: For k mismatches the data structure uses space $O(n + d \log^k d)$, is built in time $O(n + d \log^{k+1} d)$, and has a query time of $O((m + log^k d) \cdot \log \log n + occ)$. The bounds for edit distance are modified as in the indexing problem.

Applications

Approximate Indexing has a wide array of applications in signal processing, computational biology, and text retrieval among others. Approximate Dictionary Matching is important in digital libraries and text retrieval systems.

Cross References

- ▶ Compressed Text Indexing
- ▶ Indexed Approximate String Matching
- ▶ Multidimensional String Matching
- ▶ Sequential Multiple String Matching
- ▶ Suffix Array Construction
- ▶ Suffix Tree Construction in Hierarchical Memory
- ▶ Suffix Tree Construction in RAM
- ▶ Text Indexing
- ▶ Two-Dimensional Pattern Indexing

Recommended Reading

1. Aho, A.V., Corasick, M.J.: Efficient string matching. Commun. ACM **18**(6), 333–340 (1975)
2. Alstrup, S., Brodal, G.S., Rauhe, T.: Pattern matching in dynamic texts. In: Proc. of Symposium on Discrete Algorithms (SODA), 2000, pp. 819–828
3. Amir, A., Farach, M., Matias, Y.: Efficient randomized dictionary matching algorithms. In: Proc. of Symposium on Combinatorial Pattern Matching (CPM), 1992, pp. 259–272
4. Amir, A., Keselman, D., Landau, G.M., Lewenstein, N., Lewenstein, M., Rodeh, M.: Indexing and dictionary matching with one error. In: Proc. of Workshop on Algorithms and Data Structures (WADS), 1999, pp. 181–192
5. Cole, R., Kopelowitz, T., Lewenstein, M.: Suffix trays and suffix trists: Structures for faster text indexing. In: Proc. of International Colloquium on Automata, Languages and Programming (ICALP), 2006, pp. 358–369
6. Cole, R., Gottlieb, L., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proc. of the Symposium on Theory of Computing (STOC), 2004, pp. 91–100
7. Farach, M., Muthukrishnan, S.: Optimal parallel dictionary matching and compression. In: Symposium on Parallel Algorithms and Architecture (SPAA), 1995, pp. 244–253
8. Ferragina, P., Luccio, F.: Dynamic dictionary matching in external memory. Inf. Comput. **146**(2), 85–99 (1998)
9. Ferragina, P., Muthukrishnan, S., deBerg, M.: Multi-method dispatching: a geometric approach with applications to string matching. In: Proc. of the Symposium on the Theory of Computing (STOC), 1999, pp. 483–491
10. Idury, R.M., Schäffer, A.A.: Dynamic dictionary matching with failure functions. In: Proc. 3rd Annual Symposium on Combinatorial Pattern Matching, 1992, pp. 273–284
11. Karkkainen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**(6), 918–936 (2006)
12. Mehlhorn, K.: Dynamic binary search. SIAM J. Comput. **8**(2), 175–198 (1979)
13. Sahinalp, S.C., Vishkin, U.: Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: Proc. of the Foundations of Computer Science (FOCS), 1996, pp. 320–328
14. Weiner, P.: Linear pattern matching algorithm. In: Proc. of the Symposium on Switching and Automata Theory, 1973, pp. 1–11

Dilation

► Geometric Spanners
► Planar Geometric Spanners

Dilation of Geometric Networks

2005; Ebbers-Baumann, Grüne, Karpinski, Klein, Kutz, Knauer, Lingas

Rolf Klein
Institute for Computer Science, University of Bonn, Bonn, Germany

Keywords and Synonyms

Detour; Spanning ratio; Stretch factor

Problem Definition

Notations

Let $G = (V, E)$ be a plane geometric network, whose vertex set V is a finite set of point sites in $\mathbb{R}^2$, connected by an edge set E of non-crossing straight line segments with endpoints in V. For two points $p \neq q \in V$ let $\xi_G(p, q)$ denote a shortest path from p to q in G. Then

$$\sigma(p, q) := \frac{|\xi_G(p, q)|}{|pq|} \qquad (1)$$

is the detour one encounters when using network G, in order to get from p to q, instead of walking straight. Here, $|.|$ denotes the Euclidean length.

The *dilation* of G is defined by

$$\sigma(G) := \max_{p \neq q \in V} \sigma(p, q)\ . \qquad (2)$$

This value is also known as the spanning ratio or the stretch factor of G. It should, however, not be confused with the geometric dilation of a network, where the points on the edges are also being considered, in addition to the vertices.

Given a finite set S of points in the plane, one would like to find a plane geometric network $G = (V, E)$ whose dilation $\sigma(G)$ is as small as possible, such that S is contained in V. The value of

$$\Sigma(S) := \inf\{\, \sigma(G);\ G = (V, E) \text{ finite plane geometric network where } S \subset V \,\}$$

is called the *dilation of point set* S. The problem is in computing, or bounding, $\Sigma(S)$ for a given set S.

Related Work

If edge crossings were allowed one could use spanners whose stretch can be made arbitrarily close to 1; see the monographs by Eppstein [6] or Narasimhan and Smid [12]. Different types of triangulations of S are known to have their stretch factors bounded from above by small constants, among them the Delaunay triangulation of stretch ≤ 2.42; see Dobkin et al. [3], Keil and Gutwin [10], and Das and Joseph [2]. Eppstein [5] has characterized all triangulations T of dilation $\sigma(T) = 1$; these triangulations are shown in Fig. 1. Trivially, $\Sigma(S) = 1$ holds for each point set S contained in the vertex set of such a triangulation T.

Key Results

The previous remark's converse turns also out to be true.

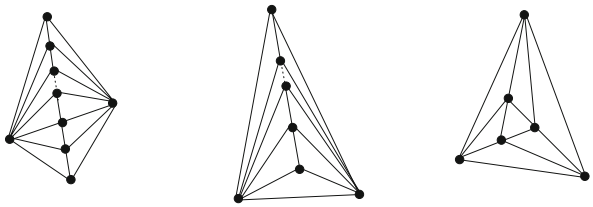
Theorem 1 ([11]) *If S is not contained in one of the vertex sets depicted in Fig. 1 then $\Sigma(S) > 1$.*

That is, if a point set S is not one of these special sets then each plane network including S in its vertex set has a dilation larger than some lower bound $1 + \eta(S)$. The proof of Theorem 1 uses the following density result. Suppose one connects each pair of points of S with a straight line segment. Let S' be the union of S and the resulting crossing points. Now the same construction is applied to S', and repeated. For the limit point set S^∞ the following theorem holds. It generalizes work by Hillar and Rhea [8] and by Ismailescu and Radoičić [9] on the intersections of lines.

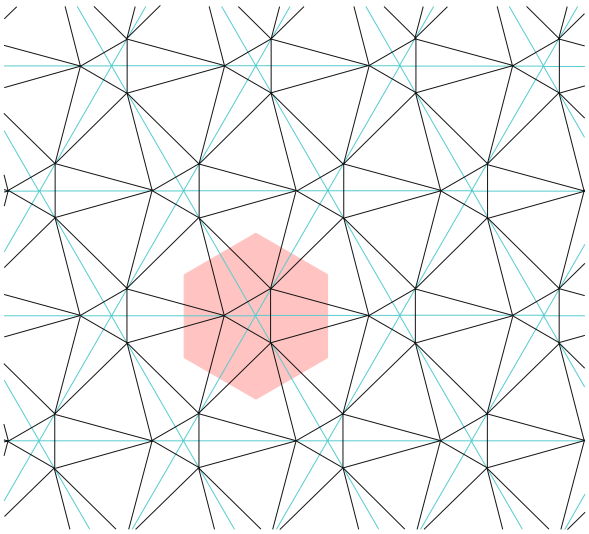
Theorem 2 ([11]) *If S is not contained in one of the vertex sets depicted in Fig. 1 then S^∞ lies dense in some polygonal part of the plane.*

For certain infinite structures can concrete lower bounds be proven.

Theorem 3 ([4]) *Let N be an infinite plane network all of whose faces have a diameter bounded from above by some constant. Then $\sigma(N) > 1.00156$ holds.*



Dilation of Geometric Networks, Figure 1
The triangulations of dilation 1



Dilation of Geometric Networks, Figure 2
A network of dilation ~ 1.1247

Theorem 4 ([4]) *Let C denote the (infinite) set of all points on a closed convex curve. Then $\Sigma(C) > 1.00157$ holds.*

Theorem 5 ([4]) *Given n families F_i, $2 \le i \le n$, each consisting of infinitely many equidistant parallel lines. Suppose that these families are in general position. Then their intersection graph G is of dilation at least $2/\sqrt{3}$.*

The proof of Theorem 5 makes use of Kronecker's theorem on simultaneous approximation. The bound is attained by the packing of equiangular triangles.

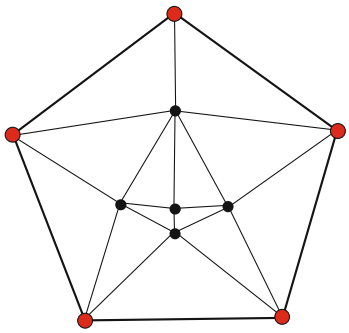
Finally, there is a general upper bound to the dilation of finite point sets.

Theorem 6 ([4]) *Each finite point set S is of dilation $\Sigma(S) < 1.1247$.*

To prove this upper bound one can embed any given finite point set S in the vertex set of a scaled, and slightly deformed, finite part of the network depicted in Fig. 2. It results from a packing of equilateral triangles by replacing each vertex with a small triangle, and by connecting neighboring triangles as indicated.

Applications

A typical university campus contains facilities like lecture halls, dorms, library, mensa, and supermarkets, which are connected by some path system. Students in a hurry are tempted to walk straight across the lawn, if the shortcut seems worth it. After a while, this causes new paths to appear. Since their intersections are frequented by many people, they attract coffee shops or other new facilities. Now, people will walk across the lawn to get quickly to a coffee shop, and so on.



Dilation of Geometric Networks, Figure 3
The best known embedding for S_5

D. Eppstein [5] has asked what happens to the lawn if this process continues. The above results show that (1) part of the lawn will be completely destroyed, and (2) the temptation to walk across the lawn cannot, in general, be made arbitrarily small by a clever path design.

Open Problems

For practical applications, upper bounds to the weight (= total edge length) of a geometric network would be valuable, in addition to upper dilation bounds. Some theoretical questions require further investigation, too. Is $\Sigma(S)$ always attained by a finite network? How to compute, or approximate, $\Sigma(S)$ for a given finite set S? Even for a set as simple as S_5, the corners of a regular 5-gon, is the dilation unknown. The smallest dilation value known, for a triangulation containing S_5 among its vertices, equals 1.0204; see Fig. 3. Finally, what is the precise value of $\sup\{\Sigma(S); S \text{ finite}\}$?

Cross References

► Geometric Dilation of Geometric Networks

Recommended Reading

1. Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Vigneron, A.: Sparse Geometric Graphs with Small Dilation. 16th International Symposium ISAAC 2005, Sanya. In: Deng, X., Du, D. (eds.) Algorithms and Computation, Proceedings. LNCS, vol. 3827, pp. 50–59. Springer, Berlin (2005)
2. Das, G., Joseph, D.: Which Triangulations Approximate the Complete Graph? In: Proc. Int. Symp. Optimal Algorithms. LNCS 401, pp. 168–192. Springer, Berlin (1989)
3. Dobkin, D.P., Friedman, S.J., Supowit, K.J.: Delaunay Graphs Are Almost as Good as Complete Graphs. Discret. Comput. Geom. **5**, 399–407 (1990)

4. Ebbers-Baumann, A., Gruene, A., Karpinski, M., Klein, R., Knauer, C., Lingas, A.: Embedding Point Sets into Plane Graphs of Small Dilation. Int. J. Comput. Geom. Appl. **17**(3), 201–230 (2007)
5. Eppstein, D.: The Geometry Junkyard. http://www.ics.uci.edu/~eppstein/junkyard/dilation-free/
6. Eppstein, D.: Spanning Trees and Spanners. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 425–461. Elsevier, Amsterdam (1999)
7. Eppstein, D., Wortman, K.A.: Minimum Dilation Stars. In: Proc. 21st ACM Symp. Comp. Geom. (SoCG), Pisa, 2005, pp. 321–326
8. Hillar, C.J., Rhea, D.L. A Result about the Density of Iterated Line Intersections. Comput. Geom.: Theory Appl. **33**(3), 106–114 (2006)
9. Ismailescu, D., Radoičić, R.: A Dense Planar Point Set from Iterated Line Intersections. Comput. Geom. Theory Appl. **27**(3), 257–267 (2004)
10. Keil, J.M., Gutwin, C.A.: The Delaunay Triangulation Closely Approximates the Complete Euclidean Graph. Discret. Comput. Geom. **7**, 13–28 (1992)
11. Klein, R., Kutz, M.: The Density of Iterated Plane Intersection Graphs and a Gap Result for Triangulations of Finite Point Sets. In: Proc. 22nd ACM Symp. Comp. Geom. (SoCG), Sedona (AZ), 2006, pp. 264–272
12. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press (2007)

Directed Perfect Phylogeny (Binary Characters)

1991; Gusfield

Jesper Jansson
Ochanomizu University, Tokyo, Japan

Keywords and Synonyms

Directed binary character compatibility

Problem Definition

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of elements called *objects*, and let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of functions from S to $\{0, 1\}$ called *characters*. For each object $s_i \in S$ and character $c_j \in C$, it is said that s_i *has* c_j if $c_j(s_i) = 1$ or that s_i *does not have* c_j if $c_j(s_i) = 0$, respectively (in this sense, characters are *binary*). Then the set S and its relation to C can be naturally represented by a matrix M of size $(n \times m)$ satisfying $M[i, j] = c_j(s_i)$ for every $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$. Such a matrix M is called a *binary character state matrix*.

Next, for each $s_i \in S$, define the set $C_{s_i} = \{c_j \in C : s_i \text{ has } c_j\}$. A *phylogeny for* S is a tree whose leaves are bijectively labeled by S, and a *directed perfect phylogeny for* (S, C) (if one exists) is a rooted phylogeny T for S in which each $c_j \in C$ is associated with exactly one edge of T in such a way that for any $s_i \in S$, the set of all characters associated with the edges on the path in T from the root to leaf s_i is equal to C_{s_i}. See Figs. 1 and 2 for two examples.

Now, define the following problem.

Problem 1 (The Directed Perfect Phylogeny Problem for Binary Characters)
INPUT: *A binary character state matrix M for some S and C.*
OUTPUT: *A directed perfect phylogeny for (S, C), if one exists; otherwise, null.*

Key Results

For the presentation below, for each $c_j \in C$, define a set $S_{c_j} = \{s_i \in S : s_i \text{ has } c_j\}$. The next lemma is the key to solving The Directed Perfect Phylogeny Problem for Binary Characters efficiently. It was first proved by Estabrook, Johnson, and McMorris [2,3], and is also known in the literature as *the pairwise compatibility theorem*. A constructive proof of the lemma can be found in, e. g., [7,11].

Lemma 1([2,3]) *There exists a directed perfect phylogeny for (S, C) if and only if for all $c_j, c_k \in C$ it holds that $S_{c_j} \cap S_{c_k} = \emptyset$, $S_{c_j} \subseteq S_{c_k}$, or $S_{c_k} \subseteq S_{c_j}$.*

Using Lemma 1, it is straightforward to construct a top-down algorithm for the problem that runs in $O(nm^2)$ time. However, a faster algorithm is possible. Gusfield [6] observed that after sorting the columns of M in non-increasing order all duplicate copies of a column appear in a consecutive block of columns and column j is to the right of column k if S_{c_j} is a proper subset of S_{c_k}, and exploited this fact together with Lemma 1 to obtain the following result:

Theorem 2 ([6]) *The Directed Perfect Phylogeny Problem for Binary Characters can be solved in $O(nm)$ time.*

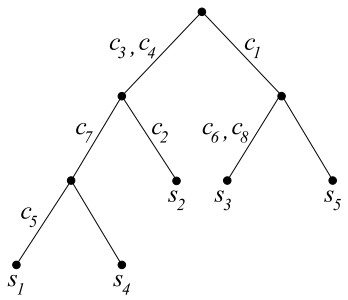
For a detailed description of the original algorithm and a proof of its correctness, see [6] or [11]. A conceptually simplified version of the algorithm based on keyword trees can be found in [7]. Gusfield [6] also gave an adversary argument to prove a corresponding lower bound of $\Omega(nm)$ on the running time, showing that his algorithm is time optimal:

Theorem 3 ([6]) *Any algorithm that decides if a given binary character state matrix M admits a directed perfect phylogeny must, in the worst case, examine all entries of M.*

Agarwala, Fernández-Baca, and Slutzki [1] noted that the input binary character state matrix is often sparse, i. e., in general, most of the objects will not have most of the characters. In addition, they noted that for the sparse case, it is more efficient to represent the input (S, C) by all the sets S_{c_j} for $j \in \{1, 2, \ldots, m\}$, where each set S_{c_j} is defined

M	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
s_1	0	0	1	1	1	0	1	0
s_2	0	1	1	1	0	0	0	0
s_3	1	0	0	0	0	1	0	1
s_4	0	0	1	1	0	0	1	0
s_5	1	0	0	0	0	0	0	0

a



b

Directed Perfect Phylogeny (Binary Characters), Figure 1
a A (5 × 8)-binary character state matrix *M*. b A directed perfect phylogeny for (*S*,*C*)

M	c_1	c_2
s_1	1	0
s_2	1	1
s_3	0	1

Directed Perfect Phylogeny (Binary Characters), Figure 2
This binary character state matrix admits no directed perfect phylogeny

as above and each S_{c_j} is specified as a linked list, than by using a binary character state matrix. Agarwala et al. [1] proved that with this alternative representation of S and C, the algorithm of Gusfield can be modified to run in time proportional to the total number of 1's in the corresponding binary character state matrix[1]:

Theorem 4 ([1]) *The variant of The Directed Perfect Phylogeny Problem for Binary Characters in which the input is given as linked lists representing all the sets S_{c_j} for $j \in \{1, 2, \ldots, m\}$ can be solved in O(h) time, where $h = \sum_{j=1}^{m} |S_{c_j}|$.*

For a description of the algorithm, refer to [1] or [5].

Applications

Directed perfect phylogenies for binary characters are used to describe the evolutionary history for a set of objects that share some observable traits and that have evolved from a "blank" ancestral object which has none of the traits. Intuitively, the root of a directed perfect phylogeny corresponds to the blank ancestral object and each directed edge $e = (u, v)$ corresponds to an evolutionary event in which the hypothesized ancestor represented by u gains the characters associated with e, transforming it into the hypothesized ancestor or object represented by v. It is assumed that each character can emerge once only during the evolutionary history and is never lost after it has been gained[2], so a leaf s_i is a descendant of the edge associated with a character c_j if and only if s_i has c_j.

Binary characters are commonly used by biologists and linguists. Traditionally, morphological traits or directly observable features of species were employed by biologists as binary characters, and recently, binary characters based on genomic information such as substrings in DNA or protein sequences, protein regulation data, and shared gaps in a given multiple alignment have become more and more prevalent. Section 17.3.2 in [7] mentions several examples where phylogenetic trees have been successfully constructed based on such types of binary character data. In the context of reconstructing the evolutionary history of natural languages, linguists often use phonological and morphological characters with just two states [9].

The Directed Perfect Phylogeny Problem for Binary Characters is closely related to *The Perfect Phylogeny Problem*, a fundamental problem in computational evolutionary biology and phylogenetic reconstruction [4,5,11]. This problem (also described in more detail in entry ► Perfect Phylogeny (Bounded Number of States)) introduces non-binary characters so that each character $c_j \in C$ has a set of allowed states $\{0, 1, \ldots, r_j - 1\}$ for some integer r_j, and for each $s_i \in S$, character c_j is in one of its allowed states. Generalizing the notation used above, define the set $S_{c_j,\alpha}$ for every $\alpha \in \{0, 1, \ldots, r_j - 1\}$ by $S_{c_j,\alpha} = \{s_i \in S :$ the state of s_i on c_j is $\alpha\}$. Then, the objective of *The Perfect Phylogeny Problem* is to construct (if possible) an *unrooted* phylogeny T for S such that the following holds: for each $c_j \in C$ and distinct states α, β of c_j,

[1]Note that Theorem 4 does not contradict Theorem 3; in fact, Gusfield's lower bound argument considers an input matrix consisting mostly of 1's.

[2]When this requirement is too strict, one can relax it to permit errors; for example, let characters be associated with more than one edge in the phylogeny (i. e., allow each character to emerge many times) but minimize the total number of associations (*Camin–Sokal optimization*), or keep the requirement that each character emerges only once but allow it to be lost multiple times (*Dollo parsimony*) [4,5]

the minimal subtree of T that connects $S_{c_j,\alpha}$ and the minimal subtree of T that connects $S_{c_j,\beta}$ are vertex-disjoint. McMorris [10] showed that the special case with $r_j = 2$ for all $c_j \in C$ can be reduced to The Directed Perfect Phylogeny Problem for Binary Characters in $O(nm)$ time (for each $c_j \in C$, if the number of 1's in column j of M is greater than the number of 0's then set entry $M[i, j]$ to $1-M[i, j]$ for all $i \in \{1, 2, \ldots, n\}$). Therefore, another application of Gusfield's algorithm [6] is as a subroutine for solving The Perfect Phylogeny Problem when $r_j = 2$ for all $c_j \in C$ in $O(nm)$ time. Even more generally, The Perfect Phylogeny Problem for directed as well as undirected *cladistic* characters can be solved in polynomial time by a similar reduction to The Directed Perfect Phylogeny Problem for Binary Characters (see [5]).

In addition to the above, it is possible to apply Gusfield's algorithm to determine whether two given trees describe compatible evolutionary history, and if so, merge them into a single tree so that no branching information is lost (see [6] for details). Finally, Gusfield's algorithm has also been used by Hanisch, Zimmer, and Lengauer [8] to implement a particular operation on documents defined in their Protein Markup Language (ProML) specification.

Cross References

- ► Perfect Phylogeny (Bounded Number of States)
- ► Perfect Phylogeny Haplotyping

Acknowledgments

Supported in part by Kyushu University, JSPS (Japan Society for the Promotion of Science), and INRIA Lille - Nord Europe.

Recommended Reading

1. Agarwala, R., Fernández-Baca, D., Slutzki, G.: Fast algorithms for inferring evolutionary trees. J. Comput. Biol. **2**, 397–407 (1995)
2. Estabrook, G.F., Johnson, C.S., Jr., McMorris, F.R.: An algebraic analysis of cladistic characters. Discret. Math. **16**, 141–147 (1976)
3. Estabrook, G.F., Johnson, C.S., Jr., McMorris, F.R.: A mathematical foundation for the analysis of cladistic character compatibility. Math. Biosci. **29**, 181–187 (1976)
4. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc., Sunderland (2004)
5. Fernández-Baca, D.: The Perfect Phylogeny Problem. In: Cheng, X., Du, D.-Z. (eds.) Steiner Trees in Industry, pp. 203–234. Kluwer Academic Publishers, Dordrecht (2001)
6. Gusfield, D.M.: Efficient algorithms for inferring evolutionary trees. Networks **21**, 19–28 (1991)
7. Gusfield, D.M.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
8. Hanisch, D., Zimmer, R., Lengauer, T.: ProML – the Protein Markup Language for specification of protein sequences, structures and families. In: Silico Biol. **2**, 0029 (2002). http://www.bioinfo.de/isb/2002/02/0029/
9. Kanj, I.A., Nakhleh, L., Xia, G.: Reconstructing evolution of natural languages: Complexity and parametrized algorithms. In: Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON 2006). Lecture Notes in Computer Science, vol. 4112, pp. 299–308. Springer, Berlin (2006)
10. McMorris, F.R.: On the compatibility of binary qualitative taxonomic characters. Bull. Math. Biol. **39**, 133–138 (1977)
11. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)

Direct Routing Algorithms

2006; Busch, Magdon-Ismail, Mavronicolas, Spirakis

Costas Busch
Department of Computer Science,
Lousiana State University, Baton Rouge, LA, USA

Keywords and Synonyms

Hot-potato routing; Bufferless packet switching; Collision-free packet scheduling

Problem Definition

The performance of a communication network is affected by the *packet collisions* which occur when two or more packets appear simultaneously in the same network node (router) and all these packets wish to follow the same outgoing link from the node. Since network links have limited available bandwidth, the collided packets wait on buffers until the collisions are resolved. Collisions cause delays in the packet delivery time and also contribute to the network performance degradation.

Direct routing is a packet delivery method which avoids packet collisions in the network. In direct routing, after a packet is injected into the network it follows a path to its destination without colliding with other packets, and thus without delays due to buffering, until the packet is absorbed at its destination node. The only delay that a packet experiences is at the source node while it waits to be injected into the network.

In order to formulate the direct routing problem, the network is modeled as a graph where all the network nodes are synchronized with a common time clock. Network links are bidirectional, and at each time step any link can be crossed by at most two packets, one packet in each direction. Given a set of packets, the *routing time* is defined to be the time duration between the first packet injection and the last packet absorbtion.

Consider a set of N packets, where each packet has its own source and destination node. In the *direct rout-*

ing problem, the goal is first to find a set of paths for the packets in the network, and second, to find appropriate injection times for the packets, so that if the packets are injected at the prescribed times and follow their paths they will be delivered to their destinations without collisions. The *direct scheduling problem* is a variation of the above problem, where the paths for the packets are given a priori, and the only task is to compute the injection times for the packets.

A *direct routing algorithm* solves the direct routing problem (similarly, a *direct scheduling algorithm* solves the direct scheduling problem). The objective of any direct algorithm is to minimize the routing time for the packets. Typically, direct algorithms are *offline*, that is, the paths and the injection schedule are computed ahead of time, before the packets are injected into the network, since the involved computation requires knowledge about all packets in order to guarantee the absence of collisions between them.

Key Results

Busch, Magdon-Ismail, Mavronicolas, and Spirakis, present in [6] a comprehensive study of direct algorithms. They study several aspects of direct routing such as the computational complexity of direct problems and also the design of efficient direct algorithms. The main results of their work are described below.

Hardness of Direct Routing

It is shown in [Sect. 4 in 6] that the optimal direct scheduling problem, where the paths are given and the objective is to compute an optimal injection schedule (that minimizes the routing time) is an NP-complete problem. This result is obtained with a reduction from vertex coloring, where vertex coloring problems are transformed to appropriate direct scheduling problems in a 2-dimensional grid. In addition, it is shown in [6] that approximations to the direct scheduling problem are as hard to obtain as approximations to vertex coloring. A natural question is what kinds of approximations can be obtained in polynomial time. This question is explored in [6] for general and specific kinds of graphs, as described below.

Direct Routing in General Graphs

A direct algorithm is given in [Section 3 in 6] that solves approximately the optimal direct scheduling problem in general network topologies. Suppose that a set of packets and respective paths are given. The injection schedule is computed in polynomial time with respect to the size of the graph and the number of packets. The routing time is measured with respect to the *congestion* C of the packet paths (the maximum number of paths that use an edge), and the *dilation* D (the maximum length of any path).

The result in [6] establishes the existence of a simple greedy direct scheduling algorithm with routing time $rt = O(C \cdot D)$. In this algorithm, the packets are processed in an arbitrary order and each packet is assigned the smallest available injection time. The resulting routing time is worst-case optimal, since there exist instances of direct scheduling problems for which no direct scheduling algorithm can achieve a better routing time. A trivial lower bound on the routing time of any direct scheduling problem is $\Omega(C + D)$, since no algorithm can deliver the packets faster than the congestion or dilation of the paths. Thus, in the general case, the algorithm in [6] has routing time $rt = O((rt^*)^2)$, where rt^* is the optimal routing time.

Direct Routing in Specific Graphs

Several direct algorithms are presented in [6] for specialized network topologies. The algorithms solve the direct routing problem where first good paths are constructed and then an efficient injection schedule is computed. Given a set of packets, let C^* and D^* denote the optimal congestion and dilation, respectively, for all possible sets of paths for the packets. Clearly, the optimal routing time is $rt^* = \Omega(C^* + D^*)$. The upper bounds in the direct algorithm in [6] are expressed in terms of this lower bound. All the algorithms run in time polynomial to the size of the input.

Tree The graph G is an arbitrary tree. A direct routing algorithm is given in [Section 3.1 in 6], where each packet follows the shortest path from its source to the destination. The injection schedule is obtained using the greedy algorithm with a particular ordering of the packets. The routing time of the algorithm is asymptotically optimal: $rt \leq 2C^* + D^* - 2 < 3 \cdot rt^*$.

Mesh The graph G is a d-dimensional mesh (grid) with n nodes [10]. A direct routing algorithm is proposed in [Section 3.2 in 6], which first constructs efficient paths for the packets with congestion $C = O(d \log n \cdot C^*)$ and dilation $D = O(d^2 \cdot D^*)$ (the congestion is guaranteed with high probability). Then, using these paths the injection schedule is computed giving a direct algorithm with the routing time:

$$rt = O(d^2 \log^2 n \cdot C^* + d^2 \cdot D^*) = O(d^2 \log^2 n \cdot rt^*)\,.$$

This result follows from a more general result which is shown in [6], that says that if the paths contain at most b "bends", i. e. at most b dimension changes, then

there is a direct scheduling algorithm with routing time $O(b \cdot C + D)$. The result follows because the constructed paths have $b = O(d \log n)$ bends.

Butterfly The graph G is a butterfly network with n input and n output nodes [10]. In [Section 3.3 in 6] the authors examine permutation routing problems in the butterfly, where each input (output) node is the source (destination) of exactly one packet. An efficient direct routing algorithm is presented in [6] which first computes good paths for the packets using Valiant's method [14,15]: two butterflies are connected back to back, and each path is formed by choosing a random intermediate node in the output of the first butterfly. The chosen paths have congestion $C = O(\lg n)$ (with high probability) and dilation $D = 2 \lg n = O(D^*)$. Given the paths, there is a direct schedule with routing time very close to optimal: $rt \leq 5 \lg n = O(rt^*)$.

Hypercube The graph G is a hypercube with n nodes [10]. A direct routing algorithm is given in [Section 3.4 in 6] for permutation routing problems. The algorithm first computes good paths for the packets by selecting a single random intermediate node for each packet. Then an appropriate injection schedule gives routing time $rt < 14 \lg n$, which is worst-case optimal since there exist permutations for which $D^* = \Omega(\lg n)$.

Lower Bound for Buffering

In [Section 5 in 6] an additional problem has been studied about the amount of buffering required to provide small routing times. It is shown in [6] that there is a direct scheduling problem for which every direct algorithm requires routing time $\Omega(C \cdot D)$; at the same time, $C + D = \Theta(\sqrt{C \cdot D}) = o(C \cdot D)$. If buffering of packets is allowed, then it is well known that there exist packet scheduling algorithms ([11,12]) with routing time very close to the optimal $O(C + D)$. In [6] it is shown that for the particular packet problem, in order to convert a direct injection schedule of routing time $O(C \cdot D)$ to a packet schedule with routing time $O(C + D)$, it is necessary to buffer packets in the network nodes in total $\Omega(N^{4/3})$ times, where a packet buffering corresponds to keeping a packet in an intermediate node buffer for a time step, and N is the number of packets.

Related Work

The only previous work which specifically addresses direct routing is for permutation problems on trees [3,13]. In these papers, the resulting routing time is $O(n)$ for any tree with n nodes. This is worst-case optimal, while the result in [6] is asymptotically optimal for all routing problems in trees.

Cypher *et al.* [7] study an online version of direct routing in which a worm (packet of length L) can be retransmitted if it is dropped (they also allow the links to have bandwidth $B \geq 1$). Adler et al. [1] study time constrained direct routing, where the task is to schedule as many packets as possible within a given time frame.They show that the time constrained version of the problem is NP-complete, and also study approximation algorithms on trees and meshes. Further, they discuss how much buffering could help in this setting.

Other models of bufferless routing are *matching routing* [2] where packets move to their destinations by swapping packets in adjacent nodes, and *hot-potato routing* [4,5,8,9] in which packets follow links that bring them closer to the destination, and if they cannot move closer (due to collisions) they are deflected toward alternative directions.

Applications

Direct routing represent collision-free communication protocols, in which packets spend the smallest amount of time possible time in the network once they are injected. This type of routing is appealing in power or resource constrained environments, such as optical networks, where packet buffering is expensive, or sensor networks where energy resources are limited. Direct routing is also important for providing quality of service in networks. There exist applications where it is desirable to provide guarantees on the delivery time of the packets after they are injected into the network, for example in streaming audio and video. Direct routing is suitable for such applications.

Cross References

- ▶ Oblivious Routing
- ▶ Packet Routing

Recommended Reading

1. Adler, M., Khanna, S., Rajaraman, R., Rosén, A.: Time-constrained scheduling of weighted packets on trees and meshes. Algorithmica **36**, 123–152 (2003)
2. Alon, N., Chung, F., Graham, R.: Routing permutations on graphs via matching. SIAM J. Discret. Math. **7**(3), 513–530 (1994)
3. Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Direct routing on trees. In: Proceedings of the Ninth Annual ACM-SIAM, Symposium on Discrete Algorithms (SODA 98), pp. 342–349. San Francisco, California, United States (1998)
4. Ben-Dor, A., Halevi, S., Schuster, A.: Potential function analysis of greedy hot-potato routing. Theor. Comput. Syst. **31**(1), 41–61 (1998)

5. Busch, C., Herlihy, M., Wattenhofer, R.: Hard-potato routing. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, pp. 278–285. Portland, Oregon, United States (2000)
6. Busch, C., Magdon-Ismail, M., Mavronicolas, M., Spirakis, P.: Direct routing: Algorithms and Complexity. Algorithmica **45**(1), 45–68 (2006)
7. Cypher, R., Meyer auf der Heide, F., Scheideler, C., Vöcking, B.: Universal algorithms for store-and-forward and wormhole routing. In: Proceedings of the 28th ACM Symposium on Theory of Computing, pp. 356–365. Philadelphia, Pennsylvania, USA (1996)
8. Feige, U., Raghavan, P.: Exact analysis of hot-potato routing. In: IEEE (ed.) Proceedings of the 33rd Annual, Symposium on Foundations of Computer Science, pp. 553–562, Pittsburgh (1992)
9. Kaklamanis, C., Krizanc, D., Rao, S.: Hot-potato routing on processor arrays. In: Proceedings of the 5th Annual ACM, Symposium on Parallel Algorithms and Architectures, pp. 273–282, Velen (1993)
10. Leighton, F.T.: Introduction to Parallel Algorithms and Architectures: Arrays – Trees – Hypercubes. Morgan Kaufmann, San Mateo (1992)
11. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and job-scheduling in O(congestion+dilation) steps. Combinatorica **14**, 167–186 (1994)
12. Leighton, T., Maggs, B., Richa, A.W.: Fast algorithms for finding O(congestion + dilation) packet routing schedules. Combinatorica **19**, 375–401 (1999)
13. Symvonis, A.: Routing on trees. Inf. Process. Lett. **57**(4), 215–223 (1996)
14. Valiant, L.G.: A scheme for fast parallel communication. SIAM J. Comput. **11**, 350–361 (1982)
15. Valiant, L.G., Brebner, G.J.: Universal schemes for parallel communication. In: Proceedings of the 13th Annual ACM, Symposium on Theory of Computing, pp. 263–277. Milwaukee, Wisconsin, United States (1981)

Distance-Based Phylogeny Reconstruction (Fast-Converging)

2003; King, Zhang, Zhou

Miklós Csűrös
Department of Computer Science, University of Montreal, Montreal, QC, Canada

Keywords and Synonyms

Learning an evolutionary tree

Problem Definition

Introduction

From a mathematical point of view, a phylogeny defines a probability space for random sequences observed at the leaves of a binary tree T. The tree T represents the unknown hierarchy of common ancestors to the sequences. It is assumed that (unobserved) ancestral sequences are associated with the inner nodes. The tree along with the associated sequences models the evolution of a molecular sequence, such as the protein sequence of a gene. In the conceptually simplest case, each tree node corresponds to a species, and the gene evolves within the organismal lineages by vertical descent.

Phylogeny reconstruction consists of finding T from observed sequences. The possibility of such reconstruction is implied by fundamental principles of molecular evolution, namely, that random mutations within individuals at the genetic level spreading to an entire mating population are not uncommon, since often they hardly influence evolutionary fitness [15]. Such mutations slowly accumulate, and, thus, differences between sequences indicate their evolutionary relatedness.

The reconstruction is theoretically feasible in several known situations. In some cases, distances can be computed between the sequences, and used in a distance-based algorithm. Such an algorithm is fast-converging if it almost surely recovers T, using sequences that are polynomially long in the size of T. Fast-converging algorithms exploit statistical concentration properties of distance estimation.

Formal Definitions

An evolutionary *topology* $U(X)$ is an unrooted binary tree in which leaves are bijectively mapped to a set of species X. A *rooted topology* T is obtained by rooting a topology U on one of the edges uv: a new node ρ is added (the *root*), the edge uv is replaced by two edges ρv and ρu, and the edges are directed outwards on paths from ρ to the leaves. The edges, vertices, and leaves of a rooted or unrooted topology T are denoted by $\mathcal{E}(T)$, $\mathcal{V}(T)$ and $\mathcal{L}(T)$, respectively.

The edges of an unrooted topology U may be equipped with a a positive *edge length* function $d\colon E(U) \mapsto (0,\infty)$. Edge lengths induce a *tree metric* $d\colon \mathcal{V}(U)\times\mathcal{V}(U)\mapsto[0,\infty)$ by the extension $d(u,v)=\sum_{e\in u\rightsquigarrow v} d(e)$, where $u \rightsquigarrow v$ denotes the unique path from u to v. The value $d(u,v)$ is called the *distance* between u and v. The pairwise distances between leaves form a *distance matrix*.

An *additive tree metric* is a function $\delta\colon X\times X\mapsto[0,\infty)$ that is equivalent to the distance matrix induced by some topology $U(X)$ and edge lengths. In certain random models, it is possible to define an additive tree metric that can be estimated from dissimilarities between sequences observed at the leaves.

In a *Markov model of character evolution* over a rooted topology T, each node u has an associated *state*, which

is a random variable $\xi(u)$ taking values over a fixed alphabet $\mathcal{A} = \{1, 2, \ldots r\}$. The vector of leaf states constitutes the *character* $\xi = \big(\xi(u)\colon u \in \mathcal{L}(T)\big)$. The states form a first-order Markov chain along every path. The joint distribution of the node states is specified by the marginal distribution of the root state, and the conditional probabilities $\mathbb{P}\{\xi(v) = b | \xi(u) = a\} = p_e(a \to b)$ on each edge e, called *edge transition probabilities*.

A *sample* of length ℓ consists of independent and identically distributed characters $\Xi = \big(\xi_i\colon i = 1, \ldots \ell\big)$. The *random sequence* associated with the leaf u is the vector $\Xi(u) = \big(\xi_i(u)\colon i = 1, \ldots \ell\big)$.

A *phylogeny reconstruction algorithm* is a function $\mathcal{F}$ mapping samples to unrooted topologies. The *success probability* is the probability that $\mathcal{F}(\Xi)$ equals the true topology.

Popular Random Models

Neyman Model [14] The edge transition probabilities are

$$p_e(a \to b) = \begin{cases} 1 - \mu_e & \text{if } a = b\,; \\ \frac{\mu_e}{r-1} & \text{if } a \neq b \end{cases}$$

with some edge-specific mutation probability $0 < \mu_e < 1 - 1/r$. The root state is uniformly distributed. A distance is usually defined by

$$d(u, v) = -\frac{r-1}{r} \ln\Big(1 - \frac{r}{r-1}\mathbb{P}\{\xi(u) \neq \xi(v)\}\Big).$$

General Markov Model There are no restrictions on the edge transition probabilities in the general Markov model. For identifiability [1,16], however, it is usually assumed that $0 < \det \mathbf{P}_e < 1$, where $\mathbf{P}_e$ is the stochastic matrix of edge transition probabilities. Possible distances in this model include the *paralinear* distance [12,1] and the *LogDet* distance [13,16]. This latter is defined by $d(u, v) = -\ln \det \mathbf{J}_{uv}$, where J_{uv} is the matrix of joint probabilities for $\xi(u)$ and $\xi(v)$.

It is often assumed in practice that sequence evolution is effected by a continuous-time Markov process operating on the edges. Accordingly, the edge length directly measures time. In particular, $\mathbf{P}_e = e^{\mathbf{Q}\cdot d(e)}$ on every edge e, where $\mathbf{Q}$ is the instantaneous rate matrix of the underlying process.

Key Results

It turns out that the hardness of reconstructing an unrooted topology U from distances is determined by its *edge depth* $\rho(U)$. Edge depth is defined as the smallest integer k for which the following holds. From each endpoint of every edge $e \in \mathcal{E}(U)$, there is a path leading to a leaf, which does not include e and has at most k edges.

Theorem 1 (Erdős, Steel, Székely, Warnow [6]) *If U has n leaves, then $\rho(U) \leq 1 + \log_2(n-1)$. Moreover, for almost all random n-leaf topologies under the uniform or Yule-Harding distributions, $\rho(U) \in O(\log \log n)$*

Theorem 2 (Erdős, Steel, Székely, Warnow [6]) *For the Neyman model, there exists a polynomial-time algorithm that has a success probability $(1 - \delta)$ for random samples of length*

$$\ell = O\Big(\frac{\log n + \log \frac{1}{\delta}}{f^2(1-2g)^{4\rho+6}}\Big), \tag{1}$$

where $0 < f = \min_e \mu_e$ and $g = \max_e \mu_e < 1/2$ are extremal edge mutation probabilities, and ρ is the edge depth of the true topology.

Theorem 2 can be extended to the general Markov model with analogous success rates for LogDet distances [7], as well as to a number of other Markov models [2].

Equation (1) shows that phylogenies can be reconstructed with high probability from polynomially long sequences. Algorithms with such sample size requirements were dubbed *fast-converging* [9]. Fast convergence was proven for the short quartet methods of Erdős et al. [6,7], and for certain variants [11] of the so-called disk-covering methods introduced by Huson et al. [9]. All these algorithms run in $\Omega(n^5)$ time. Csürös and Kao [3] initiated the study of computationally efficient fast-converging algorithms, with a cubic-time solution. Csürös [2] gave a quadratic-time algorithm. King et al. [10] designed an algorithm with an optimal running time of $O(n \log n)$ for producing a phylogeny from a matrix of estimated distances.

The short quartet methods were revisited recently: [4] described an $O(n^4)$-time method that aims at succeeding even if only a short sample is available. In such a case, the algorithm constructs a forest of "trustworthy" edges that match the true topology with high probability.

All known fast-converging distance-based algorithms have essentially the same sample bound as in (1), but Daskalakis et al. [5] recently gave a twist to the notion of fast convergence. They described a polynomial-time algorithm, which outputs the true topology almost surely from a sample of size $O(\log n)$, given that edge lengths are not too large. Such a bound is asymptotically optimal [6]. Interestingly, the sample size bound does not involve exponential dependence on the edge depth: the algorithm does not rely on a distance matrix.

Applications

Phylogenies are often constructed in molecular evolution studies, from aligned DNA or protein sequences. Fast-converging algorithms have mostly a theoretical appeal at this point. Fast convergence promises a way to handle the increasingly important issue of constructing large-scale phylogenies: see, for example, the CIPRES project (http://www.phylo.org/).

Cross References

Similar algorithmic problems are discussed under the heading ► Distance-based phylogeny reconstruction (optimal radius).

Recommended Reading

Joseph Felsenstein wrote a definitive guide [8] to the methodology of phylogenetic reconstruction.

1. Chang, J.T.: Full reconstruction of Markov models on evolutionary trees: identifiability and consistency. Math. Biosci. **137**, 51–73 (1996)
2. Csürös, M.: Fast recovery of evolutionary trees with thousands of nodes. J. Comput. Biol. **9**(2), 277–297 (2002) Conference version at RECOMB 2001
3. Csürös, M., Kao, M.-Y.: Provably fast and accurate recovery of evolutionary trees through Harmonic Greedy Triplets. SIAM J. Comput. **31**(1), 306–322 (2001) Conference version at SODA (1999)
4. Daskalakis, C., Hill, C., Jaffe, A., Mihaescu, R., Mossel, E., Rao, S.: Maximal accurate forests from distance matrices. In: Proc. Research in Computational Biology (RECOMB), pp. 281–295 (2006)
5. Daskalakis, C., Mossel, E., Roch, S.: Optimal phylogenetic reconstruction. In: Proc. ACM Symposium on Theory of Computing (STOC), pp. 159–168 (2006)
6. Erdős, P.L., Steel, M.A., Székely, L.A., Warnow, T.J.: A few logs suffice to build (almost) all trees (I). Random Struct. Algorithm **14**, 153–184 (1999) Preliminary version as DIMACS TR97-71
7. Erdős, P.L., Steel, M.A., Székely, L. A., Warnow, T.J.: A few logs suffice to build (almost) all trees (II). Theor. Comput. Sci. **221**, 77–118 (1999) Preliminary version as DIMACS TR97-72
8. Felsenstein, J.: Inferring Pylogenies. Sinauer Associates, Sunderland, Massachusetts (2004)
9. Huson, D., Nettles, S., Warnow, T.: Disk-covering, a fast converging method of phylogenetic reconstruction. J. Comput. Biol. **6**(3–4) 369–386 (1999) Conference version at RECOMB (1999)
10. King, V., Zhang, L., Zhou, Y.: On the complexity of distance-based evolutionary tree reconstruction. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 444–453 (2003)
11. Lagergren, J.: Combining polynomial running time and fast convergence for the disk-covering method. J. Comput. Syst. Sci. **65**(3), 481–493 (2002)
12. Lake, J.A.: Reconstructing evolutionary trees from DNA and protein sequences: paralinear distances. Proc. Natl. Acad. Sci. USA **91**, 1455–1459 (1994)
13. Lockhart, P.J., Steel, M.A., Hendy, M.D., Penny, D.: Recovering evolutionary trees under a more realistic model of sequence evolution. Mol. Biol. Evol. **11**, 605–612 (1994)
14. Neyman, J.: Molecular studies of evolution: a source of novel statistical problems. In: Gupta, S.S., Yackel, J. (eds) Statistical Decision Theory and Related Topics, pp. 1–27. Academic Press, New York (1971)
15. Ohta, T.: Near-neutrality in evolution of genes and gene regulation. Proc. Natl. Acad. Sci. USA **99**, 16134–16137 (2002)
16. Steel, M.A.: Recovering a tree from the leaf colourations it generates under a Markov model. Appl. Math. Lett. **7**, 19–24 (1994)

Distance-Based Phylogeny Reconstruction (Optimal Radius)

1999; Atteson
2005; Elias, Lagergren

Richard Desper[1], Olivier Gascuel[2]

[1] Department of Biology, University College London, London, UK

[2] LIRMM, National Scientific Research Center, Montpellier, France

Keywords and Synonyms

Phylogeny reconstruction; Distance methods; Performance analysis; Robustness; Safety radius approach; Optimal radius

Problem Definition

A phylogeny is an evolutionary tree tracing the shared history, including common ancestors, of a set of extant taxa. Phylogenies have been historically reconstructed using character-based (parsimony) methods, but in recent years the advent of DNA sequencing, along with the development of large databases of molecular data, has led to more involved methods. Sophisticated techniques such as likelihood and Bayesian methods are used to estimate phylogenies with sound statistical justifications. However, these statistical techniques suffer from the discrete nature of tree topology space. Since the number of tree topologies increases exponentially as a function of the number of taxa, and each topology requires separate likelihood calculation, it is important to restrict the search space and to design efficient heuristics. Distance methods for phylogeny reconstruction serve this purpose by inferring trees in a fraction of the time required for the more statistically rigorous methods. They allow dealing with thousands of taxa, while the current implementations of statistical approaches are limited to a few hundreds, and distance methods also provide fairly accurate starting trees to be further refined by more sophisticated methods. Moreover,

the input of distance methods is the matrix of pairwise evolutionary distances among taxa, which are estimated by maximum likelihood, so that distance methods also have sound statistical justifications.

Mathematically, a phylogenetic tree is a triple $T = (V, E, l)$ where V is the set of nodes representing extant taxa and ancestral species, E is the set of edges (branches), and l is a function that assigns positive lengths to each edge in E. Evolution proceeds through the tree structure as a stochastic process with a finite state space corresponding to the DNA bases or amino acids present in the DNA or protein sequences, respectively.

Any phylogenetic tree T defines a metric D_T on its leaf set $L(T)$: let $P_T(u, v)$ define the unique path through T from u to v, then the distance from u to v is set to $D_T(u, v) = \sum_{e \in P_T(u,v)} l(e)$.

Distance methods for phylogeny reconstruction rely on the observation [13] that the map $T \to D_T$ is reversible; i. e., a tree T can be reconstructed from its tree metric. While in practice D_T is not known, by using models of evolution (e. g. [10], reviewed in [5]) one can use molecular sequence data to estimate a distance matrix D that approximates D_T. As the amount of sequence data increases, the consistency of the various models of sequence evolution implies that D should converge to D_T. Thus for a distance method to be *consistent*, it is necessary that for any tree T, and for distance matrices D "close enough" to D_T, the algorithm will output T.

The present chapter deals with the question of when any distance algorithm for phylogeny reconstruction can be guaranteed to output the correct phylogeny as a function of the divergence between the metric underlying the true phylogeny and the metric estimated from the data. Atteson [1] demonstrated that this consistency can be shown for Neighbor Joining (NJ) [11], the most popular distance method, and a number of NJ's variants.

The Neighbor Joining (NJ) Algorithm of Saitou and Nei (1987)

NJ is *agglomerative*: it works by using the input matrix D to identify a pair of taxa $x, y \in L$ that are neighbors in T, i. e. there exists a node $u \in V$ such that $\{(u, x), (u, y)\} \subset E$. The algorithm creates a node c that is connected to x and y, extends the distance matrix to c, and then solves the reduced problem on $L \cup \{c\} \backslash \{x, y\}$. The pair (x, y) is chosen to minimize the following sum:

$$S_D(x, y) = (|L| - 2) \cdot D(x, y) - \sum_{z \in L} \big(D(z, x) + D(z, y)\big) .$$

The soundness of NJ is based on the observation that, if $D = D_T$ for a tree T, the value $S_D(x, y)$ will be minimized for a pair (x, y) that are neighbors in T. A number of papers (reviewed in [8]) have been dedicated to the various interpretations and properties of the S_D criterion.

The Fast Neighbor Joining (FNJ) Algorithm of Elias and Lagergren (2005)

NJ requires $\Omega(n^3)$ computations, where n is the number of taxa in the data set. Since a distance matrix only has n^2 entries, many attempts have been made to construct a distance algorithm that would only require $O(n^2)$ computations while retaining the accuracy of NJ. To this end, the best result so far is the Fast Neighbor Joining (FNJ) algorithm of Elias and Lagergren [4].

Most of the computation of NJ is used in the recalculations of the sums $S_D(x, y)$ after each agglomeration step. Although each recalculation can be performed in constant time, the number of such pairs is $\Omega(k^2)$ when k nodes are left to agglomerate, and thus, summing over k, $\Omega(n^3)$ computations are required in all.

Elias and Lagergren take a related approach to agglomeration, which does not exhaustively seek the minimum value of $S_D(x, y)$ at each step, but instead uses a heuristic to maintain a list of candidates of "visible pairs" (x, y) for agglomeration. At the $(n - k)^{th}$ step, when two neighbors are agglomerated from a k-taxa tree to form a (k-1)-taxa tree, FNJ has a list of $O(k)$ visible pairs for which $S_D(x, y)$ is calculated. The pair joined is selected from this list. By trimming the number of pairs considered, Elias and Lagergren achieved an algorithm which requires only $O(n^2)$ computations.

Safety Radius-Based Performance Analysis (Atteson 1999)

Short branches in a phylogeny are difficult to resolve, especially when they are nested deep within a tree, because relatively few mutations occurring on a short branch as opposed to on much longer pendant branches, which hides phylogenetic signal. One is faced with the choice between leaving certain evolutionary relationships unresolved (i. e., having an internal node with degree > 3), or examining when confidence can be had in the resolution of a short internal edge.

A natural formulation [9] of this question is: how long must be molecular sequences before one can have confidence in an algorithm's ability to reconstruct T accurately? An alternative formulation [1] appropriate for distance methods: if D is a distance matrix that approximates a tree metric D_T, can one have some confidence in an algorithm's ability to reconstruct T given D, based on some measure of the distance between D and D_T? For two matri-

ces, D_1 and D_2, the L_∞ distance between them is defined by $\|D_1 - D_2\|_\infty = \max_{i,j} |D_1(i,j) - D_2(i,j)|$. Moreover, let $\mu(T)$ denote the length of the shortest internal edge of a tree T.

The latter formulation leads to a definition: The *safety radius* of an algorithm $\mathcal{A}$ is the greatest value of r with the property that: given any phylogeny T, and any distance matrix D satisfying $\|D - D_T\|_\infty < r \cdot \mu(T)$, $\mathcal{A}$ will return the tree T.

Key Results

Atteson [1] answered the second question affirmatively, with two theorems.

Theorem 1 *The safety radius of NJ is* 1/2.

Theorem 2 *For no distance algorithm $\mathcal{A}$ is the safety radius of $\mathcal{A}$ greater than* 1/2.

Indeed, given any μ, one can find two different trees T_1, T_2 and a distance matrix D such that $\mu = \mu(T_1) = \mu(T_2)$ and $\|D - D_{T_1}\|_\infty = \mu/2 = \|D - D_{T_2}\|_\infty$. Since D is equidistant from two distinct tree metrics, no algorithm could assign it to the "closest" tree.

In their presentation of an optimally fast version of the NJ algorithm, Elias and Lagergren updated Atteson's results for the FNJ algorithm. They showed

Theorem 3 *The safety radius of FNJ is* 1/2.

Elias and Lagergren showed that if D is a distance matrix and D_T is a tree metric with $\|D - D_T\|_\infty < \mu(T)/2$, then FNJ will output the same tree (T) as NJ.

Applications

Phylogeny is a quite active field within evolutionary biology and bioinformatics. As more proteins and DNA sequences become available, the need for fast and accurate phylogeny estimation algorithms is ever increasing as phylogeny not only serves to reconstruct species history but also to decipher genomes. To date, NJ remains one of the most popular algorithms for phylogeny building, and is by far the most popular of the distance methods, with well over 1000 citations per year.

Open Problems

With increasing amounts of sequence data becoming available for an increasing number of species, distance algorithms such as NJ should be useful for quite some time. Currently, the bottleneck in the process of building phylogenies is not the problem of searching topology space, but rather the problem of building distance matrices. The brute force method to build a distance matrix on n taxa from sequences with l positions requires $\Omega(ln^2)$ computations, and typically $l \gg n$. Elias and Lagergren proposed an $\Omega(ln^{1.376})$ algorithm based on Hamming distance and matrix calculations. However, this algorithm only applies to over-simple distance estimators [10]. Extending this result to more realistic models would be a great advance.

A number of distance-based tree building algorithms have been analyzed in the safety radius framework. Atteson [1] dealt with a large class of neighbor joining-like algorithms, and Gascuel and McKenzie [7] studied the ultrametric setting where the correct tree T is rooted and all tree leaves are at the same distance from the root. Such trees are very common; they are called "molecular clock" trees in phylogenetics and "indexed hierarchies" in data analysis. In this setting, the optimal safety radius is equal to 1 (instead of 1/2) and a number of standard algorithms (e. g. UPGMA, with time complexity in $O(n^2)$) have a safety radius of 1. However, experimental studies (see below) showed that not all algorithms with optimal safety radius achieve the same accuracy, indicating that the safety radius approach should be sharpened to provide better theoretical analysis of method performance.

Experimental Results

Computer simulation is the most standard way to assess algorithm accuracy in phylogenetics. A tree is randomly generated as well as a sequence at tree root, whose evolution is simulated along the tree edges. A reconstruction algorithm is tested using the sequences observed at the tree leaves, thus mimicking the phylogenetic task. Various measures exist to compare the correct and the inferred trees, and algorithm performance is assessed as the average measure over repeated experiments. Elias and Lagergren [4] showed that FNJ (in $O(n^2)$) is just slightly outperformed by NJ (in $O(n^3)$), while numerous simulations (e. g. [3,12]) indicated that NJ is beaten by more recent algorithms (all in $O(n^3)$ or less), namely BioNJ [6], WEIGHBOR [2], FastME [3] and STC [12].

Data Sets

A large number of data sets is stored by the TreeBASE project, at http://www.treebase.org.

URL to Code

For a list of leading phylogeny packages, see Joseph Felsenstein's website at http://evolution.genetics.washington.edu/phylip/software.html

Cross References

- ▶ Approximating Metric Spaces by Tree Metrics
- ▶ Directed Perfect Phylogeny (Binary Characters)
- ▶ Distance-Based Phylogeny Reconstruction (Fast-Converging)
- ▶ Perfect Phylogeny (Bounded Number of States)
- ▶ Perfect Phylogeny Haplotyping
- ▶ Phylogenetic Tree Construction from a Distance Matrix

Recommended Reading

1. Atteson, K.: The performance of neighbor-joining methods of phylogenetic reconstruction. Algorithmica **25**, 251–278 (1999)
2. Bruno, W.J., Socci, N.D., Halpern, A.L.: Weighted Neighbor Joining: A Likelihood-Based Approach to Distance-Based Phylogeny Reconstruction. Mol. Biol. Evol. **17**, 189–197 (2000)
3. Desper, R., Gascuel, O.: Fast and Accurate Phylogeny Reconstruction Algorithms Based on the Minimum – Evolution Principle. J. Comput. Biol. **9**, 687–706 (2002)
4. Elias, I. Lagergren, J.: Fast Neighbor Joining. In: Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP), pp. 1263–1274 (2005)
5. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Sunderland, Massachusetts (2004)
6. Gascuel, O.: BIONJ: an Improved Version of the NJ Algorithm Based on a Simple Model of Sequence Data. Mol. Biol. Evol. **14**, 685–695 (1997)
7. Gascuel, O. McKenzie, A.: Performance Analysis of Hierarchical Clustering Algorithms. J. Classif. **21**, 3–18 (2004)
8. Gascuel, O., Steel, M.: Neighbor-Joining Revealed. Mol. Biol. Evol. **23**, 1997–2000 (2006)
9. Huson, D.H., Nettles, S., Warnow, T.: Disk-covering, a fast-converging method for phylogenetic tree reconstruction. J. Comput. Biol. **6**, 369–386 (1999)
10. Jukes, T.H., Cantor, C.R.: Evolution of Protein Molecules. In: Munro, H.N. (ed.), Mammalian Protein Metabolism, pp. 21–132, Academic Press, New York (1969)
11. Saitou, N., Nei, M.: The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees. Mol. Biol. Evol. **4**, 406–425 (1987)
12. Vinh, L.S., von Haeseler, A.: Shortest triplet clustering: reconstructing large phylogenies using representative sets. BMC Bioinformatics **6**, 92 (2005)
13. Zarestkii, K.: Reconstructing a tree from the distances between its leaves. Uspehi Mathematicheskikh Nauk **20**, 90–92 (1965) (in russian)

Distributed Algorithms for Minimum Spanning Trees

1983; Gallager, Humblet, Spira

Sergio Rajsbaum
Math Institute, National Autonomous University of Mexico, Mexico City, Mexico

Keywords and Synonyms

Minimum weight spanning tree

Problem Definition

Consider a communication network, modeled by an undirected weighted graph $G = (V, E)$, where $|V| = n$, $|E| = m$. Each vertex of V represents a processor of unlimited computational power; the processors have unique identity numbers (ids), and they communicate via the edges of E by sending messages to each other. Also, each edge $e \in E$ has associated a weight $w(e)$, known to the processors at the endpoints of e. Thus, a processor knows which edges are incident to it, and their weights, but it does not know any other information about G. The network is *asynchronous:* each processor runs at an arbitrary speed, which is independent of the speed of other processors. A processor may wake up spontaneously, or when it receives a message from another processor. There are no failures in the network. Each message sent arrives at its destination within a finite but arbitrary delay. A *distributed algorithm* A for G is a set of local algorithms, one for each processor of G, that include instructions for sending and receiving messages along the edges of the network. Assuming that A terminates (i. e. all the local algorithms eventually terminate), its *message complexity* is the total number of messages sent over any execution of the algorithm, in the worst case. Its *time complexity* is the worst case execution time, assuming processor steps take negligible time, and message delays are normalized to be at most 1 unit.

A *minimum spanning tree* (MST) of G is a subset E' of E such that the graph $T = (V, E')$ is a tree (connected and acyclic) and its total weight, $w(E') = \sum_{e \in E'} w(e)$ is as small as possible. The computation of an MST is a central problem in combinatorial optimization, with a rich history dating back to 1926 [2], and up to now; the book [12] collects properties, classical results, applications, and recent research developments.

In the *distributed MST problem* the goal is to design a distributed algorithm A that terminates always, and computes an MST T of G. At the end of an execution, each processor knows which of its incident edges belong to the tree T and which not (i. e. the processor writes in a local output register the corresponding incident edges). It is remarkable that in the distributed version of the MST problem, a communication network is solving a problem where the input is the network itself. This is one of the fundamental starting points of network algorithms.

It is not hard to see that if all edge weights are different, the MST is unique. Due to the assumption that

processors have unique ids, it is possible to assume that all edge weights are different: whenever two edge weights are equal, ties are broken using the processor ids of the edge endpoints. Having a unique MST facilitates the design of distributed algorithms, as processors can locally select edges that belong to the unique MST. Notice that if processors do not have unique ids, and edge weights are not different, there is no deterministic MST (nor any spanning tree) distributed algorithm, because it may be impossible to break the symmetry of the graph, for example, in the case it is a cycle with all edge weights equal.

Key Results

The distributed MST problem has been studied since 1977, and dozens of papers have been written on the subject. In 1983, the fundamental distributed *GHS algorithm* in [5] was published, the first to solve the MST problem with $O(m + n \log n)$ message complexity. The paper has had a very significant impact on research in distributed computing and won the 2004 Edsger W. Dijkstra Prize in Distributed Computing.

It is not hard to see that any distributed MST algorithm must have $\Omega(m)$ message complexity (intuitivelly, at least one message must traverse each edge). Also, results in [3,4] imply an $\Omega(n \log n)$ message complexity lower bound for the problem. Thus, the GHS algorithm is optimal in terms of message complexity.

The $\Omega(m + n \log n)$ message complexity lower bound for the construction of an MST applies also to the problem of finding an arbitrary spanning tree of the graph. However, for specific graph topologies, it may be easier to find an arbitrary spanning tree than to find an MST. In the case of a complete graph, $\Omega(n^2)$ messages are necessary to construct an MST [8], while an arbitrary spanning tree can be constructed in $O(n \log n)$ messages [7].

The time complexity of the GHS algorithm is $O(n \log n)$. In [1] it is described how to improve its time complexity to $O(n)$, while keeping the optimal $O(m + n \log n)$ message complexity. It is clear that $\Omega(D)$ time is necessary for the construction of a spanning tree, where D is the diameter of the graph. And in the case of an MST the time complexity may depend on other parameters of the graph. For example, due to the need for information flow among processors residing on a common cycle, as in an MST construction, at least one edge of the cycle must be excluded from the MST. If messages of unbounded size are allowed, an MST can be easily constructed in $O(D)$ time, by collecting the graph topology and edge weights in a root processor. The problem becomes interesting in the more realistic model where messages are of size $O(\log n)$, and an edge weight can be sent in a single message. When the number of messages is not important, one can assume without loss of generality that the model is synchronous. For near time optimal algorithms and lower bounds see [10] and references herein.

Applications

The distributed MST problem is important to solve, both theoretically and practically, as an MST can be used to save on communication, in various tasks such as broadcast and leader election, by sending the messages of such applications over the edges of the MST.

Also, research on the MST problem, and in particular the MST algorithm of [5], has motivated a lot of work. Most notably, the algorithm of [5], introduced various techniques that have been in widespread use for multicasting, query and reply, cluster coordination and routing, protocols for handshake, synchronization, and distributed phases. Although the algorithm is intuitive and is easy to comprehend, it is sufficiently complicated and interesting that it has become a challenge problem for formal verification methods e. g. [11].

Open Problems

There are many open problems in this area, only a few significant ones are mentioned. As far as message complexity, although the asymptotically tight bound of $O(m + n \log n)$ for the MST problem in general graphs is known, finding the actual constants remains an open problem. There are smaller constants known for general spanning trees than for MST though [6].

As mentioned above, near time optimal algorithms and lower bounds appear in [10] and references herein. The optimal time complexity remains an open problem. Also, in a synchronous model for overlay networks, where all processors are directly connected to each other, an MST can be constructed in sublogarithmic time, namely $O(\log \log n)$ communication rounds [9], and no corresponding lower bound is known.

Cross References

► Synchronizers, Spanners

Recommended Reading

1. Awerbuch, B.: Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In: Proc. of the 19th Annual ACM Symposium on Theory of Computing, pp. 230–240. ACM, USA (1987)

2. Borůvka, O.: Otakar Borůvka on minimum spanning tree problem (translation of both the 1926 papers, comments, history). Disc. Math. **233**, 3–36 (2001)
3. Burns, J.E.: A formal model for message-passing systems. Indiana University, Bloomington, TR-91, USA (1980)
4. Frederickson, G., Lynch, N.: The impact of synchronous communication on the problem of electing a leader in a ring. In: Proc. of the 16th Annual ACM Symposium on Theory of Computing, pp. 493–503. ACM, USA (1984)
5. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Prog. Lang. Systems **5**(1), 66–77 (1983)
6. Johansen, K.E., Jorgensen, U.L., Nielsen, S.H.: A distributed spanning tree algorithm. In: Proc. 2nd Int. Workshop on Distributed Algorithms (DISC). Lecture Notes in Computer Science, vol. 312, pp. 1–12. Springer, Berlin Heidelberg (1987)
7. Korach, E., Moran, S., Zaks, S.: Tight upper and lower bounds for some distributed algorithms for a complete network of processors. In: Proc. 3rd Symp. on Principles of Distributed Computing (PODC), pp. 199–207. ACM, USA (1984)
8. Korach, E., Moran, S., Zaks, S.: The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. In: Proc. 4th Symp. on Principles of Distributed Computing (PODC), pp. 277–286. ACM, USA (1985)
9. Lotker, Z., Patt-Shamir, B., Pavlov, E., Peleg, D.: Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. SIAM J. Comput. **35**(1), 120–131 (2005)
10. Lotker, Z., Patt-Shamir, B., Peleg, D.: Distributed MST for constant diameter graphs. Distrib. Comput. **18**(6), 453–460 (2006)
11. Moses, Y., Shimony, B.: A new proof of the GHS minimum spanning tree algorithm. In: Distributed Computing, 20th Int. Symp. (DISC), Stockholm, Sweden, September 18–20, 2006. Lecture Notes in Computer Science, vol. 4167, pp. 120–135. Springer, Berlin Heidelberg (2006)
12. Wu, B.Y., Chao, K.M.: Spanning Trees and Optimization Problems (Discrete Mathematics and Its Applications). Chapman Hall, USA (2004)

Distributed Computing

Distributed Vertex Coloring

2004; Finocchi, Panconesi, Silvestri

Devdatt Dubhashi
Department of Computer Science, Chalmers University of Technology and Gothenburg University, Gothenburg, Sweden

Keywords and Synonyms

Vertex coloring; Distributed computation

Problem Definition

The *vertex coloring problem* takes as input an undirected graph $G := (V, E)$ and computes a *vertex coloring*, i. e. a function, $c : V \to [k]$ for some positive integer k such that adjacent vertices are assigned different colors (that is, $c(u) \neq c(v)$ for all $(u, v) \in E$). In the $(\Delta + 1)$ vertex coloring problem, k is set equal to $\Delta + 1$ where Δ is the maximum degree of the input graph G. In general, $(\Delta + 1)$ colors could be necessary as the example of a clique shows. However, if the graph satisfies certain properties, it may be possible to find colorings with far fewer colors. Finding the minimum number of colors possible is a computationally hard problem: the corresponding decision problems are NP-complete [5]. In Brooks–Vizing colorings, the goal is to try to find colorings that are near optimal.

In this paper, the model of computation used is the synchronous, message passing framework as used in standard distributed computing [11]. The goal is then to describe very simple algorithms that can be implemented easily in this distributed model that simultaneously are *efficient* as measured by the number of rounds required and have good performance *quality* as measured by the number of colors used. For efficiency, the number of rounds is require to be poly-logarithmic in n, the number of vertices in the graph and for performance quality, the number of colors used is should be near-optimal.

Key Results

Key theoretical results related to distributed $(\Delta + 1)$-vertex coloring are due to Luby [9] and Johansson [7]. Both show how to compute a $(\Delta + 1)$-coloring in $O(\log n)$ rounds with high probability. For Brooks–Vizing colorings, Kim [8] showed that if the graph is square or triangle free, then it is possible to color it with $O(\Delta/\log \Delta)$ colors. If, moreover, the graph is regular of sufficiently high degree ($\Delta \gg \lg n$), then Grable and Panconesi [6] show how to color it with $O(\Delta/\log \Delta)$ colors in $O(\log n)$ rounds. See [10] for a comprehensive discussion of probabilistic techniques to achieve such colorings.

The present paper makes a comprehensive experimental analysis of distributed vertex coloring algorithms of the kind analyzed in these papers on various classes of graphs. The results are reported in Sect. “Experimental Results” below and the data sets used are described in Sect. “Data Sets”.

Applications

Vertex coloring is a basic primitive in many applications: classical applications are *scheduling problems* involving a number of pairwise restrictions on which jobs can be done simultaneously. For instance, in attempting to schedule classes at a university, two courses taught by the same faculty member cannot be scheduled for the same time slot. Similarly, two course that are required by the same group of students also should not conflict. The problem of determining the minimum number of time slots needed subject to these restrictions can be cast as a vertex coloring problem. One very active application for vertex coloring is *register allocation*. The register allocation problem is to assign variables to a limited number of hardware registers during program execution. Variables in registers can be accessed much quicker than those not in registers. Typically, however, there are far more variables than registers so it is necessary to assign multiple variables to registers. Variables conflict with each other if one is used both before and after the other within a short period of time (for instance, within a subroutine). The goal is to assign variables that do not conflict so as to minimize the use of non-register memory. A simple approach to this is to create a graph where the nodes represent variables and an edge represents conflict between its nodes. A coloring is then a conflict-free assignment. If the number of colors used is less than the number of registers then a conflict-free register assignment is possible. Modern applications include *assigning frequencies* to mobile radios and other users of the electro-magnetic spectrum. In the simplest case, two customers that are sufficiently close must be assigned different frequencies, while those that are distant can share frequencies. The problem of minimizing the number of frequencies is then a vertex coloring problem For more applications and references, see Michael Trick's coloring page [12].

Open Problems

The experimental analysis shows convincingly and rather surprisingly that the simplest, trivial, version of the algorithm actually performs best uniformly! In particular,it significantly outperforms the algorithms which have been analyzed rigorously. The authors give some heuristic recurrences that describe the performance of the trivial algorithm. It is a challenging and interesting open problem to give a rigorous justification of these recurrences. Alternatively, and less appealing, a rigorous argument that shows that the trivial algorithm dominates the ones analyzed by Luby and Johansson is called for. Other issues about how local structure of the graph impacts on the performance of such algorithms (which is hinted at in the paper) is worth subjecting to further experimental and theoretical analysis.

Experimental Results

All the algorithms analyzed start by assigning an initial *palette* of colors to each vertex, and then repeating the following simple iteration round:

1. *Wake up!*: Each vertex independently of the others wakes up with a certain probability to participate in the coloring in this round.
2. *Try!*: Each vertex independently of the others, selects a tentative color from its palette of colors at this round.
3. *Resolve conflicts!*: If no neighbor of a vertex selects the same tentative color, then this color becomes final. Such a vertex exits the algorithm, and the remaining vertices update their palettes accordingly. If there is a conflict, then it is resolved in one of two ways: Either all conflicting vertices are deemed unsuccessful and proceed to the next round, or an independent set is computed, using the so-called Hungarian heuristic, amongst all the vertices that chose the same color. The vertices in the independent set receive their final colors and exit. The Hungarian heuristic for independent set is to consider the vertices in random order, deleting all neighbors of an encountered vertex which itself is added to the independent set, see [1, p. 91] for a cute analysis of this heuristic to prove Turan's Theorem.
4. *Feed the Hungry!*: If a vertex runs out of colors in its palette, then fresh new colors are given to it.

Several parameters can be varied in this basic scheme: the wake up probability, the conflict resolution and the size of the initial palette are the most important ones.

In $(\Delta + 1)$-coloring, the initial palette for a vertex v is set to $[\Delta] := \{1, \cdots, \Delta + 1\}$ (global setting) or $[d(v) + 1]$ (where $d(v)$ is the degree of vertex v) (local setting). The experimental results indicate that (a) the best wake-up probability is 1, (b) the local palette version is as good as the global one in running time, but can achieve significant color savings and (c) the Hungarian heuristic can be used with vertex identities rather than random numbers giving good results.

In the Brooks–Vizing colorings, the initial palette is set to $[d(v)/s]$ where s is a *shrinking factor*. The experimental results indicate that uniformly, the best algorithm is the one where the wake-up probability is 1, and conflicts are resolved by the Hungarian heuristic. This is both with respect to the running time, as well as the number of colors used. Realistically useful values of s are between 4 and 6 resulting in Δ/s-colorings. The running time performance is excellent, with even graphs with a thousand vertices col-

ored within 20–30 rounds. When compared to the best sequential algorithms, these algorithms use between twice or thrice as many colors, but are much faster.

Data Sets

Test data was both generated synthetically using various random graph models, and benchmark real life test sets from the second DIMACS implementation challenge [3] and Joe Culberson's web-site [2] were also used.

Cross References

- ▶ Graph Coloring
- ▶ Randomization in Distributed Computing
- ▶ Randomized Gossiping in Radio Networks

Recommended Reading

1. Alon, N., Spencer, J.: The Probabilistic Method. Wiley (2000)
2. Culberson, J.C.: http://web.cs.ualberta.ca/~joe/Coloring/index.html
3. Ftp site of DIMACS implementation challenges, ftp://dimacs.rutgers.edu/pub/challenge/
4. Finocchi, I., Panconesi, A., Silvestri, R.: An experimental Analysis of Simple Distributed Vertex Coloring Algorithms. Algorithmica **41**, 1–23 (2004)
5. Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman (1979)
6. Grable, D.A., Panconesi, A.: Fast distributed algorithms for Brooks–Vizing colorings. J. Algorithms **37**, 85–120 (2000)
7. Johansson, Ö.: Simple distributed $(\Delta + 1)$-coloring of graphs. Inf. Process. Lett. **70**, 229–232 (1999)
8. Kim, J.-H.: On Brook's Theorem for sparse graphs. Combin. Probab. Comput. **4**, 97–132 (1995)
9. Luby, M.: Removing randomness in parallel without processor penalty. J. Comput. Syst. Sci. **47**(2), 250–286 (1993)
10. Molly, M., Reed, B.: Graph Coloring and the Probabilistic method. Springer (2002)
11. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. In: SIAM Monographs on Discrete Mathematics and Applications 5 (2000)
12. Trick, M.: Michael Trick's coloring page: http://mat.gsia.cmu.edu/COLOR/color.html

Dominating Set

- ▶ Data Reduction for Domination in Graphs
- ▶ Greedy Set-Cover Algorithms

Dynamic Problems

- ▶ Fully Dynamic Connectivity
- ▶ Robust Geometric Computation
- ▶ Voltage Scheduling

Dynamic Trees

2005; Tarjan, Werneck

Renato F. Werneck
Microsoft Research Silicon Valley, La Avenida, CA, USA

Keywords and Synonyms

Link-cut trees

Problem Definition

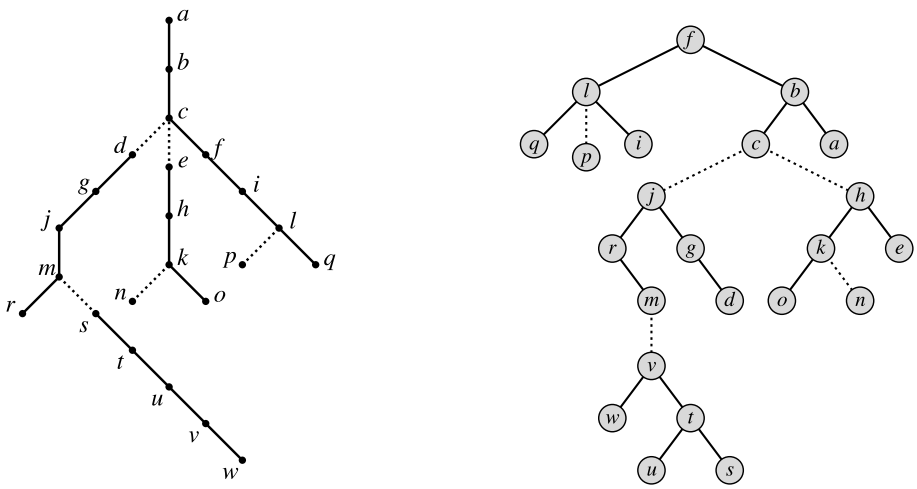
The *dynamic tree problem* is that of maintaining an arbitrary n-vertex forest that changes over time through edge insertions (*links*) and deletions (*cuts*). Depending on the application, one associates information with vertices, edges, or both. Queries and updates can deal with individual vertices or edges, but more commonly they refer to entire paths or trees. Typical operations include finding the minimum-cost edge along a path, determining the minimum-cost vertex in a tree, or adding a constant value to the cost of each edge on a path (or of each vertex of a tree). Each of these operations, as well as *links* and *cuts*, can be performed in $O(\log n)$ time with appropriate data structures.

Key Results

The obvious solution to the dynamic tree problem is to represent the forest explicitly. This, however, is inefficient for queries dealing with entire paths or trees, since it would require actually traversing them. Achieving $O(\log n)$ time per operation requires mapping each (possibly unbalanced) input tree into a balanced tree, which is better suited to maintaining information about paths or trees implicitly. There are three main approaches to perform the mapping: path decomposition, tree contraction, and linearization.

Path Decomposition

The first efficient dynamic tree data structure was Sleator and Tarjan's *ST-trees* [13,14], also known as *link-cut trees* or simply *dynamic trees*. They are meant to represent rooted trees, but the user can change the root with the *evert* operation. The data structure partitions each input tree into vertex-disjoint paths, and each path is represented as a binary search tree in which vertices appear in symmetric order. The binary trees are then connected according to how the paths are related in the forest. More precisely, the root of a binary tree becomes a *middle child* (in the data structure) of the parent (in the forest) of the topmost



Dynamic Trees, Figure 1
An ST-tree (adapted from [14]). On the *left*, the original tree, rooted at *a* and already partitioned into paths; on the *right*, the actual data structure. *Solid edges* connect nodes on the same path; *dashed edges* connect different paths

vertex of the corresponding path. Although a node has no more than two children (left and right) within its own binary tree, it may have arbitrarily many middle children. See Fig. 1. The path containing the root (*qlifcba* in the example) is said to be *exposed*, and is represented as the topmost binary tree. All path-related queries will refer to this path. The *expose* operation can be used to make any vertex part of the exposed path.

With standard balanced binary search trees (such as red-black trees), ST-trees support each dynamic tree operation in $O(\log^2 n)$ amortized time. This bound can be improved to $O(\log n)$ amortized with locally biased search trees, and to $O(\log n)$ in the worst case with globally biased search trees. Biased search trees (described in [5]), however, are notoriously complicated. A more practical implementation of ST-trees uses *splay trees*, a self-adjusting type of binary search trees, to support all dynamic tree operations in $O(\log n)$ amortized time [14].

Tree Contraction

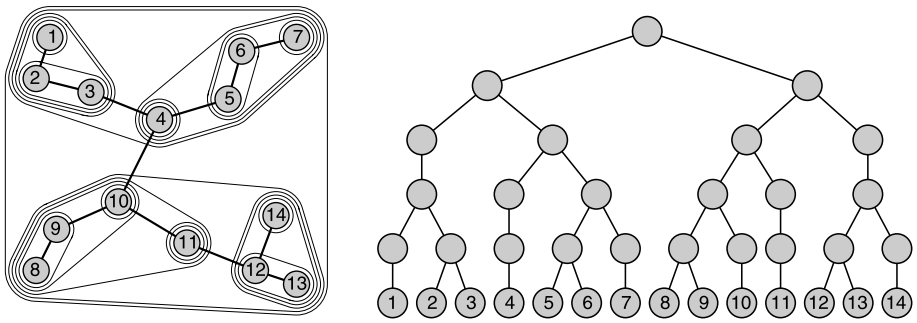
Unlike ST-trees, which represent the input trees directly, Frederickson's *topology trees* [6,7,8] represent a *contraction* of each tree. The original vertices constitute level 0 of the contraction. Level 1 represents a partition of these vertices into *clusters*: a degree-one vertex can be combined with its only neighbor; vertices of degree two that are adjacent to each other can be clustered together; other vertices are kept as singletons. The end result will be a smaller tree, whose own partition into clusters yields level 2. The process is repeated until a single cluster remains. The topology tree is a representation of the contraction, with each cluster having as children its constituent clusters on the level below. See Fig. 2.

With appropriate pieces of information stored in each cluster, the data structure can be used to answer queries about the entire tree or individual paths. After a *link* or *cut*, the affected topology trees can be rebuilt in $O(\log n)$ time.

The notion of tree contraction was developed independently by Miller and Reif [11] in the context of parallel algorithms. They propose two basic operations, *rake* (which eliminates vertices of degree one) and *compress* (which eliminates vertices of degree two). They show that $O(\log n)$ rounds of these operations are sufficient to contract any tree to a single cluster. Acar et al. translated a variant of their algorithm into a dynamic tree data structure, *RC-trees* [1], which can also be seen as a randomized (and simpler) version of topology trees.

A drawback of topology trees and RC-trees is that they require the underlying forest to have vertices with bounded (constant) degree in order to ensure $O(\log n)$ time per operation. Similarly, although ST-trees do not have this limitation when aggregating information over paths, they require bounded degrees to aggregate over trees. Degree restrictions can be addressed by "ternarizing" the input forest (replacing high-degree vertices with a series of low-degree ones [9]), but this introduces a host of special cases.

Alstrup et al.'s *top trees* [3,4] have no such limitation, which makes them more generic than all data structures previously discussed. Although also based on tree con-

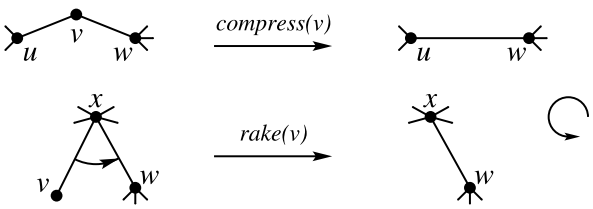


Dynamic Trees, Figure 2
A topology tree (adapted from [7]). On the *left*, the original tree and its multilevel partition; on the *right*, a corresponding topology tree

traction, their clusters behave not like vertices, but like *edges*. A *compress* cluster combines two edges that share a degree-two vertex, while a *rake* cluster combines an edge with a degree-one endpoint with a second edge adjacent to its other endpoint. See Fig. 3.

Top trees are designed so as to completely hide from the user the inner workings of the data structure. The user only specifies what pieces of information to store in each cluster, and (through call-back functions) how to update them after a cluster is created or destroyed when the tree changes. As long as the operations are properly defined, applications that use top trees are completely independent of how the data structure is actually implemented, i. e., of the order in which *rakes* and *compresses* are performed.

In fact, top trees were not even proposed as stand-alone data structures, but rather as an interface on top of topology trees. For efficiency reasons, however, one would rather have a more direct implementation. Holm, Tarjan, Thorup and Werneck have presented a conceptually simple stand-alone algorithm to update a top tree after a *link* or *cut* in $O(\log n)$ time in the worst case [17]. Tarjan and Werneck [16] have also introduced *self-adjusting top trees*, a more efficient implementation of top trees based on path decomposition: it partitions the input forest into edge-disjoint paths, represents these paths as splay trees, and connects these trees appropriately. Internally, the data structure is very similar to ST-trees, but the paths are edge-disjoint (instead of vertex-disjoint) and the ternarization step is incorporated into the data structure itself. All the user sees, however, are the *rakes* and *compresses* that characterize tree contraction.



Dynamic Trees, Figure 3
The *rake* and *compress* operations, as used by top trees (from [16])

Linearization

ET-trees, originally proposed by Henzinger and King [10] and later slightly simplified by Tarjan [15], use yet another approach to represent dynamic trees: *linearization*. It maintains an *Euler tour* of the each input tree, i. e., a closed path that traverses each edge twice—once in each direction. The tour induces a linear order among the vertices and arcs, and therefore can be represented as a balanced binary search tree. Linking and cutting edges from the forest corresponds to joining and splitting the affected binary trees, which can be done in $O(\log n)$ time. While linearization is arguably the simplest of the three approaches, it has a crucial drawback: because each edge appears twice, the data structure can only aggregate information over trees, not paths.

Lower Bounds

Dynamic tree data structures are capable of solving the *dynamic connectivity* problem on acyclic graphs: given two vertices v and w, decide whether they belong to the same tree or not. Pătraşcu and Demaine [12] have proven a lower bound of $\Omega(\log n)$ for this problem, which is matched by the data structures presented here.

Applications

Sleator and Tarjan's original application for dynamic trees was Dinic's blocking flow algorithm [13]. Dynamic trees

are used to maintain a forest of arcs with positive residual capacity. As soon as the source s and the sink t become part of the same tree, the algorithm sends as much flow as possible along the s-t path; this reduces to zero the residual capacity of at least one arc, which is then *cut* from the tree. Several maximum flow and minimum-cost flow algorithms incorporating dynamic trees have been proposed ever since (some examples are [9,15]). Dynamic tree data structures, especially those based on tree contraction, are also commonly used within dynamic graph algorithms, such as the dynamic versions of minimum spanning trees [6,10], connectivity [10], biconnectivity [6], and bipartiteness [10]. Other applications include the evaluation of dynamic expression trees [8] and standard graph algorithms [13].

Experimental Results

Several studies have compared the performance of different dynamic-tree data structures; in most cases, ST-trees implemented with splay trees are the fastest alternative. Frederickson, for example, found that topology trees take almost 50% more time than splay-based ST-trees when executing dynamic tree operations within a maximum flow algorithm [8]. Acar et al. [2] have shown that RC-trees are significantly slower than splay-based ST-trees when most operations are *links* and *cuts* (such as in network flow algorithms), but faster when queries and value updates are dominant. The reason is that splaying changes the structure of ST-trees even during queries, while RC-trees remain unchanged.

Tarjan and Werneck [17] have presented an experimental comparison of several dynamic tree data structures. For random sequences of *links* and *cuts*, splay-based ST-trees are the fastest alternative, followed by splay-based ET-trees, self-adjusting top trees, worst-case top trees, and RC-trees. Similar relative performance was observed in more realistic sequences of operations, except when queries far outnumber structural operations; in this case, the self-adjusting data structures are slower than RC-trees and worst-case top trees. The same experimental study also considered the "obvious" implementation of ST-trees, which represents the forest explicitly and require linear time per operation in the worst case. Its simplicity makes it significantly faster than the $O(\log n)$-time data structures for path-related queries and updates, unless paths are hundred nodes long. The sophisticated solutions are more useful when the underlying forest has high diameter or there is a need to aggregate information over trees (and not only paths).

Cross References

- ▶ Fully Dynamic Connectivity
- ▶ Fully Dynamic Connectivity: Upper and Lower Bounds
- ▶ Fully Dynamic Higher Connectivity
- ▶ Fully Dynamic Higher Connectivity for Planar Graphs
- ▶ Fully Dynamic Minimum Spanning Trees
- ▶ Fully Dynamic Planarity Testing
- ▶ Lower Bounds for Dynamic Connectivity
- ▶ Routing

Recommended Reading

1. Acar, U.A., Blelloch, G.E., Harper, R., Vittes, J.L., Woo, S.L.M.: Dynamizing static algorithms, with applications to dynamic trees and history independence. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 524–533. SIAM (2004)
2. Acar, U.A., Blelloch, G.E., Vittes, J.L.: An experimental analysis of change propagation in dynamic trees. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 41–54 (2005)
3. Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Minimizing diameters of dynamic trees. In: Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP), Bologna, Italy, 7–11 July 1997. Lecture Notes in Computer Science, vol. 1256, pp. 270–280. Springer (1997)
4. Alstrup, S., Holm, J., Thorup, M., de Lichtenberg, K.: Maintaining information in fully dynamic trees with top trees. ACM Trans. Algorithms **1**(2), 243–264 (2005)
5. Bent, S.W., Sleator, D.D., Tarjan, R.E.: Biased search trees. SIAM J. Comput. **14**(3), 545–568 (1985)
6. Frederickson, G.N.: Data structures for on-line update of minimum spanning trees, with applications. SIAM J. Comput. **14**(4), 781–798 (1985)
7. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. SIAM J. Comput. **26**(2), 484–538 (1997)
8. Frederickson, G.N.: A data structure for dynamically maintaining rooted trees. J. Algorithms **24**(1), 37–65 (1997)
9. Goldberg, A.V., Grigoriadis, M.D., Tarjan, R.E.: Use of dynamic trees in a network simplex algorithm for the maximum flow problem. Math. Progr. **50**, 277–290 (1991)
10. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarihmic time per operation. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC), pp. 519–527 (1997)
11. Miller, G.L., Reif, J.H.: Parallel tree contraction and its applications. In: Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 478–489 (1985)
12. Pătraşcu, M., Demaine, E.D.: Lower bounds for dynamic connectivity. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), pp. 546–553 (2004)
13. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**(3), 362–391 (1983)
14. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**(3), 652–686 (1985)

15. Tarjan, R.E.: Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. Math. Prog. **78**, 169–177 (1997)
16. Tarjan, R.E., Werneck, R.F.: Self-adjusting top trees. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 813–822 (2005)
17. Tarjan, R.E., Werneck, R.F.: Dynamic trees in practice. In: Proceedings of the 6th Workshop on Experimental Algorithms (WEA). Lecture Notes in Computer Science, vol. 4525, pp. 80–93 (2007)
18. Werneck, R.F.: Design and Analysis of Data Structures for Dynamic Trees. Ph. D. thesis, Princeton University (2006)