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.*

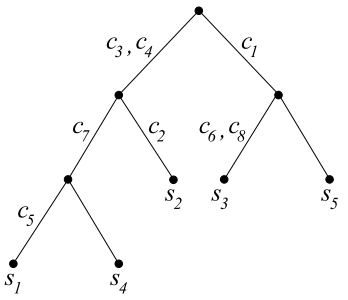
For a detailed description of the original algorithm and a proof of its correctness, see [6] or [11]. A conceptually simplified version of the algorithm based on keyword trees can be found in [7]. Gusfield [6] also gave an adversary argument to prove a corresponding lower bound of $\Omega(nm)$ on the 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 $S_{c_j,\alpha} = \{s_i \in S : \text{the state of } s_i \text{ on } c_j \text{ is } \alpha\}$. 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-*