Perceptron Algorithm, Table 1

Online Perceptron	Kernel-based Online Perceptron
INITIALIZATION: $\mathbf{w}_1 = \mathbf{0}$	INITIALIZATION: $\mathbf{I}_1 = \{\cdot\}$
For $t = 1, 2, \ldots$	For $t = 1, 2, \ldots$
Receive an instance $\mathbf{x}_t$	Receive an instance $\mathbf{x}_t$
Predict an outcome $\hat{y}_t = \text{sign}(\langle \mathbf{w}_t, \mathbf{x}_t \rangle)$	Predict an outcome $\hat{y}_t = \text{sign}\left(\sum_{i \in I_t} K(\mathbf{x}_i, \mathbf{x}_t)\right)$
Receive correct outcome $y_t \in \{+1, -1\}$	Receive correct outcome $y_t \in \{+1, -1\}$
Update: $\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t & \text{if } \hat{y}_t \neq y_t \\ \mathbf{w}_t & \text{otherwise} \end{cases}$	Update: $I_{t+1} = \begin{cases} I_t \cup \{t\} & \text{if } \hat{y}_t \neq y_t \\ I_t & \text{otherwise} \end{cases}$

small, then $f(\mathbf{x})$ is likely to perform well on unseen examples as well.

Finally, it should be noted that the Perceptron algorithm was used for other purposes such as solving linear programming [3] and training support vector machines [14]. In addition, variants of the Perceptron was used for numerous additional problems such as online learning on a budget [8,4], multiclass categorization and ranking problems [6,7], and discriminative training for hidden Markov models [5].

Cross References

► Support Vector Machines

Recommended Reading

1. Agmon., S.: The relaxation method for linear inequalities. Can. J. Math. **6**(3), 382–392 (1954)
2. Block., H. D.: The perceptron: A model for brain functioning. Rev. Mod. Phys. **34**, 123–135 (1962)
3. Blum, A., Dunagan J. D.: Smoothed analysis of the perceptron algorithm for linear programming. In: SODA, (2002)
4. Cesa-Bianchi, N., Gentile, C.: Tracking the best hyperplane with a simple budget perceptron. In: Proceedings of the Nineteenth Annual Conference on Computational Learning Theory, (2006)
5. Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: Conference on Empirical Methods in Natural Language Processing, (2002)
6. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive aggressive algorithms. J. Mach. Learn. Res. **7** (2006)
7. Crammer, K., Singer, Y.: A new family of online algorithms for category ranking. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (2002)
8. Dekel, O., Shalev-Shwartz, S., Singer, Y.: The Forgetron: A kernel-based perceptron on a fixed budget. In: Advances in Neural Information Processing Systems 18 (2005)
9. Freund, Y., Schapire, R. E.: Large margin classification using the perceptron algorithm. In: Proceedings of the Eleventh Annual Conference on Computational Learning Theory (1998)
10. Gentile, C.: The robustness of the p-norm algorithms. Mach. Learn. **53**(3) (2002)
11. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. The MIT Press, (1969)
12. Novikoff, A. B. J.: On convergence proofs on perceptrons. In: Proceedings of the Symposium on the Mathematical Theory of Automata, volume XII, pp. 615–622, (1962)
13. Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386–407 (1958)
14. Shalev-Shwartz, S., Singer, Y.: A new perspective on an old perceptron algorithm. In: Proceedings of the Eighteenth Annual Conference on Computational Learning Theory, (2005)
15. Vapnik, V. N.: Statistical Learning Theory. Wiley (1998)

Perfect Phylogeny (Bounded Number of States)

1997; Kannan, Warnow

JESPER JANSSON
Ochanomizu University, Tokyo, Japan

Keywords and Synonyms

Compatibility of characters with a bounded number of states; Convex tree-realization of partitions containing a bounded number of sets

Problem Definition

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of elements called *objects* and *species*, and let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of functions called *characters* such that each $c_j \in C$ is a function from S to the set $\{0, 1, \ldots, r_j - 1\}$ for some integer r_j. For every $c_j \in C$, the set $\{0, 1, \ldots, r_j - 1\}$ is called the set of *allowed states* of character c_j, and for any $s_i \in S$ and $c_j \in C$, it is said that the *state of* s_i *on* c_j is α, or that the *state of* c_j *for* s_i is α, where $\alpha = c_j(s_i)$. The *character state matrix* for S and C is the $(n \times m)$-matrix in which entry (i, j) for any

$i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$ equals the state of s_i on c_j.

In this chapter, a *phylogeny for* S is an unrooted tree whose leaves are bijectively labeled by S. For every $c_j \in C$ and $\alpha \in \{0, 1, \ldots, r_j - 1\}$, define the set $S_{c_j,\alpha}$ by $S_{c_j,\alpha} = \{s_i \in S :$ the state of s_i on c_j is $\alpha\}$. A *perfect phylogeny for* (S, C) (if one exists) is a phylogeny T for S such that the following holds: for each $c_j \in C$ and pair of allowed states α, β of c_j with $\alpha \neq \beta$, 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. See Fig. 1 for an example. *The Perfect Phylogeny Problem* is the following:

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

Below, define $r = \max_{j \in \{1,2,\ldots,m\}} r_j$.

Key Results

The following negative result was proved by Bodlaender, Fellows, and Warnow [2] and, independently, by Steel [13]:

Theorem 1 ([2,13]) *The Perfect Phylogeny Problem is NP-hard.*

On the other hand, certain restrictions of The Perfect Phylogeny Problem can be solved efficiently. One important special case occurs if the number of allowed states of each character is limited[1]. For this case, Agarwala and Fernández-Baca [1] designed a dynamic programming-based algorithm that builds perfect phylogenies on certain subsets of S called *c-clusters* (also referred to as *proper clusters* in [5,10] and *character subfamilies* in [6]) in a bottom-up fashion. Each c-cluster G has the property that: (1) G and $S \setminus G$ share at most one state of each character; and (2) for at least one character, G and $S \setminus G$ share no states. The number of c-clusters is at most $2^r m$, and the algorithm's total running time is $O(2^{3r}(nm^3 + m^4))$, i. e., exponential in r. (Hence, The Perfect Phylogeny Problem is polynomial-time solvable if the number of allowed states of every character is upper-bounded by $O(\log(m + n))$.) Subsequently, Kannan and Warnow [10] presented a modified algorithm with improved running time. They restructured the algorithm of [1] to eliminate one of the three nested loops that steps through all possible c-clusters and added a pre-processing step which speeds up the innermost loop. The resulting time complexity is given by:

Perfect Phylogeny (Bounded Number of States), Table 1
The running times of the fastest known algorithms for The Perfect Phylogeny Problem with a bounded number of states

r	Running time	Reference
2	$O(nm)$	[11] together with [7]
3	$\min\{O(nm^2), O(n^2m)\}$	[3,10] together with [9]
4	$\min\{O(nm^2), O(n^2m)\}$	[10] together with [9]
≥ 5	$O(2^{2r}nm^2)$	[10]

Theorem 2 ([10]) *The algorithm of Kannan and Warnow in [10] solves The Perfect Phylogeny Problem in* $O(2^{2r}nm^2)$ *time.*

A perfect phylogeny T for (S, C) is called *minimal* if no tree which results by contracting an edge of T is a perfect phylogeny for (S, C). In [10], Kannan and Warnow also showed how to extend their algorithm to enumerate all minimal perfect phylogenies for (S, C) by constructing a directed acyclic graph that implicitly stores the set of all perfect phylogenies for (S, C).

Theorem 3 ([10]) *The extended algorithm of Kannan and Warnow in [10] enumerates the set of all minimal perfect phylogenies for (S, C) so that the maximum computation time between two consecutive outputs is* $O(2^{2r}nm^2)$.

For very small values of r, even faster algorithms are known. Refer to the table in Table 1 for a summary. If $r = 2$ then the problem can be solved in $O(nm)$ time by reducing it to The *Directed* Perfect Phylogeny Problem for Binary Characters (see, e. g., Encyclopedia ▶ Directed Perfect Phylogeny (Binary Characters) for a definition of this variant of the problem) using $O(nm)$ time [7,11] and then applying Gusfield's $O(nm)$-time algorithm [7]. If $r = 3$ or $r = 4$, the problem is solvable in $O(n^2m)$ time by another algorithm by Kannan and Warnow [9], which is faster than the algorithm from Theorem 2 when $n < m$. Also note that for the case $r = 3$, there exists an older algorithm by Dress and Steel [3] whose running time coincides with that of the algorithm in Theorem 2.

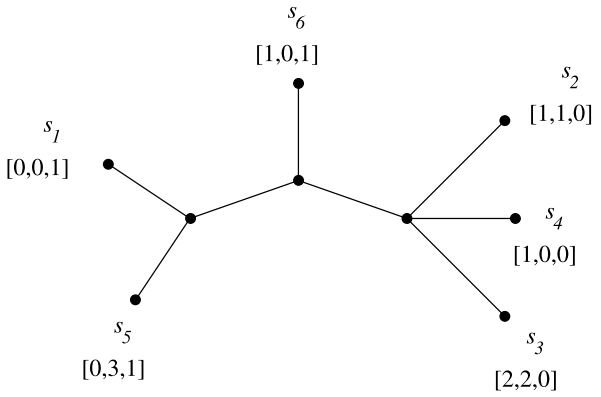
Applications

A central goal in computational evolutionary biology and phylogenetic reconstruction is to develop efficient methods for constructing, from some given data, a phylogenetic tree that accurately describes the evolutionary relationships among a set of objects (e. g., biological species or

[1] For other variants of The Perfect Phylogeny Problem which can be solved efficiently, see, for example, entries ▶ Directed Perfect Phylogeny (Binary Characters) of this Encyclopedia or the survey by Fernández-Baca [5] .

M	c_1	c_2	c_3
s_1	0	0	1
s_2	1	1	0
s_3	2	2	0
s_4	1	0	0
s_5	0	3	1
s_6	1	0	1

(a)



(b)

Perfect Phylogeny (Bounded Number of States), Figure 1
a An example of a character state matrix M for $S = \{s_1, s_2, \ldots, s_6\}$ and $C = \{c_1, c_2, c_3\}$ with $r_1 = 3$, $r_2 = 4$, and $r_3 = 2$, i. e., $r = 4$. b A perfect phylogeny for (S, C). For convenience, the states of all three characters for each object are shown

other taxa, populations, proteins, genes, natural languages, etc.) believed to have been produced by an evolutionary process. One of the most widely used techniques for reconstructing a phylogenetic tree is to represent the objects as vectors of character states and look for a tree that clusters objects which have a lot in common. The Perfect Phylogeny Problem can be regarded as the ideal special case of this approach in which the given data contains no errors, evolution is tree-like, and each character state can emerge only once in the evolutionary history.

However, data obtained experimentally seldom admits a perfect phylogeny, so various optimization versions of the problem such as *maximum parsimony* and *maximum compatibility* are often considered in practice; as might be expected, these strategies generally lead to NP-complete problems, but there exist many heuristics that work well for most inputs. See, e. g. [4,5,12], for a further discussion and references. Nevertheless, algorithms for The Perfect Phylogeny Problem may be useful even when the data does not admit a perfect phylogeny, for example if there exists a perfect phylogeny for $m - O(1)$ of the characters in C. In fact, in one crucial step of their proposed character-based methodology for determining the evolutionary history of a set of related natural languages, Warnow, Ringe, and Taylor [14] consider all subsets of C in decreasing order of cardinality, repeatedly applying the algorithm of [10] until a largest subset of C which admits a perfect phylogeny is found. The ideas behind the algorithms of [1] and [10] have also been utilized and extended by Fernández-Baca and Lagergren [6] in their algorithm for computing *near-perfect phylogenies* in which the constraints on the output have been relaxed in order to permit non-perfect phylogenies whose so-called penalty score is less than or equal to a prespecified parameter q (see [6] for details).

The motivation for considering a bounded number of states is that characters based on directly observable traits are, by the way they are defined, naturally bounded by some small number (often 2). When biomolecular data is used to define characters, the number of allowed states is typically bounded by a constant; e. g., $r = 2$ for SNP markers, $r = 4$ for DNA or RNA sequences, or $r = 20$ for amino-acid sequences (see also Encyclopedia ► Directed Perfect Phylogeny (Binary Characters)). Moreover, characters with $r = 2$ can be useful in comparative linguistics [8].

Open Problems

An important open problem is to determine whether the running time of the algorithm of Kannan and Warnow [10] can be improved. As pointed out in [5], it would be especially interesting to find out if The Perfect Phylogeny Problem is solvable in $O(2^{2r} nm)$ time for any r, or more generally, in $O(f(r) \cdot nm)$ time, where f is a function of r which does not depend on n or m, since this would match the fastest known algorithm for the special case $r = 2$ (see Table 1). Another open problem is to establish lower bounds on the computational complexity of The Perfect Phylogeny Problem with a bounded number of states.

Cross References

► Directed Perfect Phylogeny (Binary Characters)
► 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.: A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. SIAM J. Comput. **23**, 1216–1224 (1994)
2. Bodlaender, H.L., Fellows, M.R., Warnow, T.: Two strikes against perfect phylogeny. In: Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP 1992). Lecture Notes in Computer Science, vol. 623, pp. 273–283. Springer, Berlin (1992)
3. Dress, A., Steel, M.: Convex tree realizations of partitions. Appl. Math. Lett. **5**, 3–6 (1992)
4. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc. Sunderland, Massachusetts (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, Netherlands (2001)
6. Fernández-Baca, D., Lagergren, J.: A polynomial-time algorithm for near-perfect phylogeny. SIAM J. Comput. **32**, 1115–1127 (2003)
7. Gusfield, D.M.: Efficient algorithms for inferring evolutionary trees. Networks **21**, 19–28 (1991)
8. 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)
9. Kannan, S., Warnow, T.: Inferring evolutionary history from DNA sequences. SIAM J. Comput. **23**, 713–737 (1994)
10. Kannan, S., Warnow, T.: A fast algorithm for the computation and enumeration of perfect phylogenies. SIAM J. Comput. **26**, 1749–1763 (1997)
11. McMorris, F.R.: On the compatibility of binary qualitative taxonomic characters. Bull. Math. Biol. **39**, 133–138 (1977)
12. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)
13. Steel, M.A.: The complexity of reconstructing trees from qualitative characters and subtrees. J. Classification **9**, 91–116 (1992)
14. Warnow, T., Ringe, D., Taylor, A.: Reconstructing the evolutionary history of natural languages. In: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96), pp. 314–322 (1996)

Perfect Phylogeny Haplotyping

2005; Ding, Filkov, Gusfield

Giuseppe Lancia
Department of Mathematics and Computer Science, University of Udine, Udine, Italy

Keywords and Synonyms

Alleles phasing

Problem Definition

In the context of the *perfect phylogeny haplotyping* (PPH) problem, each vector $h \in \{0,1\}^m$ is called a *haplotype*, while each vector $g \in \{0,1,2\}^m$ is called a *genotype*. Haplotypes are binary encodings of DNA sequences, while genotypes are ternary encodings of *pairs* of DNA sequences (one sequence for each of the two homologous copies of a certain chromosome).

Two haplotypes h' and h'' are said to *resolve* a genotype g if, at each position j: (i) if $g_j \in \{0,1\}$ then both $h'_j = g_j$ and $h''_j = g_j$; (ii) if $g_j = 2$ then either $h'_j = 0$ and $h''_j = 1$ or $h'_j = 1$ and $h''_j = 0$. If h' and h'' resolve g, we write $g = h' + h''$. An instance of the PPH problem consists of a set $G = \{g^1, g^2, \ldots, g^n\}$ of genotypes. A set H of haplotypes such that, for each $g \in G$, there are $h', h'' \in H$ with $g = h' + h''$, is called a *resolving set* for G.

A *perfect phylogeny* for a set H of haplotypes is a rooted tree T for which

- the set of leaves is H and the root is labeled by some binary vector r;
- each index $j \in \{1, \ldots, m\}$ labels exactly one edge of T;
- if an edge e is labeled by an index k, then, for each leaf h that can be reached from the root via a path through e, it is $h_k \neq r_k$.

Without loss of generality, it can be assumed that the vector labeling the root is $r = 0$. Within the PPH problem, T is meant to represent the evolution of the sequences at the leaves from a common ancestral sequence (the root). Each edge labeled with an index represents a point in time when a mutation happened at a specific site. This model of evolution is also known as *coalescent* [11]. It can be shown that a perfect phylogeny for H exists if and only if for all choices of four haplotypes $h^1, \ldots, h^4 \in H$ and two indices i, j,

$$\{h^a_i\, h^a_j\,, 1 \leq a \leq 4\} \neq \{00, 01, 10, 11\}\,.$$

Given the above definitions, the problem surveyed in this entry is the following:

Perfect Phylogeny Haplotyping Problem (PPH): Given a set G of genotypes, find a resolving set H of haplotypes and a perfect phylogeny T for H, or determine that such a resolving set does not exist.

In a slightly different version of the above problem, one may require to find all perfect phylogenies for H instead of just one (in fact, all known algorithms for PPH do find all perfect phylogenies).

The perfect phylogeny problem was introduced by Gusfield [7], who also proposed a nearly linear-time $O(nm\,\alpha(nm))$-algorithm for its solution (where $\alpha()$ is the extremely slowly growing inverse Ackerman function).