

5

Parametric Inference

Radu Mihaescu

Graphical models are powerful statistical tools that have been applied to a wide variety of problems in computational biology: sequence alignment, ancestral genome reconstruction, etc. A graphical model consists of a graph whose vertices have associated random variables representing biological objects, such as entries in a DNA sequence, and whose edges have associated parameters that model transition or dependence relations between the random variables at the nodes. In many cases we will know the contents of only a subset of the model vertices, the *observed random variables*, and nothing about the contents of the remaining ones, the *hidden random variables*. A common example is a phylogenetic tree on a set of current species with given DNA sequences, but with no information about the DNA of their extinct ancestors. The task of finding the most likely set of values of the hidden random variables (also known as the *explanation*) given the set of observed random variables and the model parameters, is known as *inference in graphical models*.

Clearly, inference drawn about the hidden data is highly dependent on the topology and parameters (transition probabilities) of the graphical model. The topology of the model will be determined by the biological process being modeled, while the assumptions one can make about the nature of evolution, site mutation and other biological phenomena, allow us to restrict the space of possible transition probabilities to certain parameterized families. This raises several questions. If our choice of parameters is slightly off, will the explanation change? What other choices of parameters will give the same explanation? Can we find all possible explanations and the sets of parameters which yield them? These are the sorts of questions we will answer in this chapter, using the tools of *parametric inference*, which solves the inference problem for all possible sets of parameters simultaneously.

We present the *polytope propagation algorithm* for parametric inference, which was first introduced in [Pachter and Sturmfels, 2004a]. This algorithm is the polytope algebra version (see Section 2.3) of a classical method in the theory of graphical models, known as *sum-product decomposition*. We examine the polytope propagation algorithm in Section 5.2, and, in particular, we describe the details of the algorithm in the context of two very important problems in computational biology: the hidden Markov model for gene annotation

and the pair hidden Markov model for genome alignment. The analysis relies heavily on the theory developed in Sections 1.4, 2.2 and 2.3.

The running time of polytope propagation is exponential in the number of parameters. Therefore, in applications where the number of parameters is very large, it is of practical interest to specialize most of them to fixed values and study the dependence of the explanation upon variations of the remaining few parameters. In Section 5.4 we give an explicit presentation of an algorithm that does this efficiently, together with an analysis of its running time. As we will see, the complexity of the algorithm is polynomial in the length of the sequences for a fixed number of unspecialized parameters.

5.1 Tropical sum-product decompositions

We begin by showing that the problem of inference for fixed parameters can be regarded as the tropical version of computing the marginal probability of the observed data. For a graphical model with hidden and observed random variables denoted by σ and τ respectively, the probability of the observed sequence τ is

$$\text{Prob}(\tau) = \sum_{\sigma} p_{\sigma,\tau}, \quad (5.1)$$

where $p_{\sigma,\tau}$ is the probability of having states σ at the hidden nodes and states τ at the observed nodes of the model. This is the probability of the observed data marginalized over all possible values for the hidden data. The task of finding an explanation corresponds to identifying the set of hidden states $\bar{\sigma}$ with maximum a posteriori probability of generating the observed data τ . In other words:

$$\bar{\sigma} = \operatorname{argmax}_{\sigma} \{p_{\sigma,\tau}\}.$$

Now following the notation of Chapter 2, let $w_* = -\ln(p_*)$. Then the above equation becomes

$$\bar{\sigma} = \operatorname{argmin}_{\sigma} \{w_{\sigma,\tau}\},$$

which is exactly the marginalization in (5.1), performed in tropical algebra:

$$w_{\bar{\sigma}} = \bigoplus_{\sigma} w_{\sigma,\tau}. \quad (5.2)$$

The reader is referred to Section 2.1 for more details on tropical algebra.

In general, marginal probabilities for acyclic graphical models can be computed in time which is polynomial in the size of the model using the *sum-product decomposition*, which is a recursive representation of a polynomial in terms of smaller polynomials. Such a decomposition is very useful for computing values of polynomial expressions with a large number of monomials, where a direct symbolic computation would be very costly. In the literature on hidden Markov models this is known as the *forward algorithm*.

As we can see from the above analysis, evaluating the marginal probability

tropically is equivalent to solving the inference problem. See also Remarks 2.17, 2.18 and 4.13. Therefore, the sum-product decomposition of marginal probabilities, when it exists, naturally yields efficient algorithms for inference with fixed parameters. In the following subsections we exemplify this with the *Viterbi algorithm* for hidden Markov models and the *Needleman–Wunsch algorithm* for sequence alignment (see Section 2.2).

5.1.1 The sum-product algorithm for HMMs

The hidden Markov model is one of the simplest and most popular models used in computational biology. In this subsection we will describe the sum-product algorithm for the HMM. We use the notation of Section 1.4, to which we also refer the reader unfamiliar with the model. Suppose that we have an HMM of length n , with hidden states σ_i , $i \in [n]$, taking values in an alphabet Σ with l letters, and observed variables τ_i , $i \in [n]$, taking values in the alphabet Σ' of size l' . The model parameters are the “transition” probability matrix $\theta \in \mathbb{R}^{l \times l}$ and the “emission” probability matrix $\theta' \in \mathbb{R}^{l \times l'}$. If one assumes a uniform initial distribution on the states of the first hidden node, the probability of occurrence of the sequence (σ, τ) is therefore

$$p_{\sigma, \tau} = \frac{1}{l} \theta'_{\sigma_1, \tau_1} \theta_{\sigma_1, \sigma_2} \theta'_{\sigma_2, \tau_2} \theta_{\sigma_2, \sigma_3} \cdots \theta'_{\sigma_n, \tau_n}.$$

The marginal probability of the observed sequence $\tau = \tau_1 \tau_2 \dots \tau_n$ is

$$p_\tau = \sum_{\sigma} p_{\sigma, \tau}. \tag{5.3}$$

By tropicalizing and maintaining the notation from the beginning of the section we see that, as before, the explanation for the sequence of observations τ is given by

$$\begin{aligned} \bar{\sigma} &= \operatorname{argmin}_{\sigma} \{w_{\sigma, \tau}\}, \\ w_{\bar{\sigma}} &= \bigoplus_{\sigma} w_{\sigma, \tau}. \end{aligned} \tag{5.4}$$

The problem of computing (5.3) can easily be solved by noticing that the probability p_τ has the following decomposition:

$$p_\tau = \sum_{\sigma_n=1}^l \theta'_{\sigma_n, \tau_n} \left(\sum_{\sigma_{n-1}=1}^l \theta_{\sigma_{n-1}, \sigma_n} \theta'_{\sigma_{n-1}, \tau_{n-1}} \left(\cdots \left(\sum_{\sigma_1=1}^l \theta_{\sigma_1, \sigma_2} \theta'_{\sigma_1, \tau_1} \right) \cdots \right) \right). \tag{5.5}$$

Computing p_τ using this decomposition is known as the *forward algorithm* for HMMs. Its time complexity is $O(l^2n)$, as can easily be checked.

Observe that tropicalizing this algorithm gives us a way of efficiently solving

equation (5.4). By taking $u_{i,j} = -\log(\theta_{i,j})$ and $v_{i,j} = -\log(\theta'_{i,j})$, we obtain

$$\begin{aligned} \bigoplus_{\sigma} (v_{\sigma_1\tau_1} \odot u_{\sigma_1\sigma_2} \odot v_{\sigma_2\tau_2} \cdots \odot u_{\sigma_{n-1}\sigma_n} \odot v_{\sigma_n\tau_n}) &= \\ \bigoplus_{\sigma_n} (v_{\sigma_n\tau_n} \odot (\bigoplus_{\sigma_{n-1}} (v_{\sigma_{n-1}\tau_{n-1}} \odot u_{\sigma_{n-1}\sigma_n} \cdots \odot (\bigoplus_{\sigma_1} (v_{\sigma_1\tau_1} \odot u_{\sigma_1\sigma_2})))))) &= \end{aligned} \tag{5.6}$$

Evaluating this quantity by recursively computing the parentheses in the above formula is known as the *Viterbi algorithm*, and has the same time complexity as its non-tropical version, the forward algorithm.

5.1.2 The sum-product algorithm for sequence alignment

The sequence alignment problem asks for the best possible alignment between two words $\sigma^1 = \sigma_1^1\sigma_2^1 \dots \sigma_n^1$ and $\sigma^2 = \sigma_1^2\sigma_2^2 \dots \sigma_n^2$ over the alphabet $\Sigma = \{\mathbf{A}, \mathbf{C}, \mathbf{G}, \mathbf{T}\}$ that have evolved from a common ancestor via insertions, deletions or mutations of sites in the genetic sequence. A full description of the problem can be found in Section 2.2. As in Section 2.2, we represent an alignment by an edit string h over the alphabet $\{H, I, D\}$ such that $\#H + \#D = n$ and $\#H + \#I = m$. Let $\mathcal{A}_{n,m}$ be the set of all edit strings.

Each element $h \in \mathcal{A}_{n,m}$ corresponds naturally to a pair of words (μ^1, μ^2) over the alphabet $\Sigma \cup \{-\}$ such that μ^1 consists of a copy of σ^1 together with inserted “-” characters, and similarly μ^2 is a copy of σ^2 with inserted “-” characters, see (2.8).

Now consider the pair hidden Markov model for sequence alignment presented in Section 2.2. Equation (2.15) gives us the marginal probability f_{σ^1, σ^2} of observing the pair of sequences σ^1 and σ^2 :

$$f_{\sigma^1, \sigma^2} = \sum_{h \in \mathcal{A}_{n,m}} \prod_{i=1}^{|h|} \theta_{\mu_i^1, \mu_i^2} \cdot \prod_{i=2}^{|h|} \theta'_{h_{i-1}, h_i} \tag{5.7}$$

Here the parameters θ and θ' are as in Section 2.2.

Just as before, we will be interested in the tropical version of the above formula, which gives the alignment with the largest *a posteriori* probability, given the parameters of the model and the observed sequences. Letting $w_{i,j} = -\ln \theta_{i,j}$ and $w'_{i,j} = -\ln \theta'_{i,j}$, equation (5.7) yields

$$\text{trop}(f_{\sigma^1, \sigma^2}) = \bigoplus_{h \in \mathcal{A}_{n,m}} \bigodot_{i=1}^{|h|} w_{\mu_i^1, \mu_i^2} \cdot \bigodot_{i=2}^{|h|} w'_{h_{i-1}, h_i} \tag{5.8}$$

The above relation computes the negative logarithm of the *maximum a posteriori* (MAP) probability over the set of possible alignments. This is equivalent to finding a minimum path in the alignment graph of Section 2.2, which can be solved through the Needleman–Wunsch algorithm, a version of the sum-product algorithm, based on the recursive decomposition of (5.8) described below.

Let $\sigma_{\leq i}^1$ denote the sequence $\sigma_1^1\sigma_2^1 \dots \sigma_i^1$. Let $\sigma_{\leq j}^2$ be defined in the same

way. Also define $\Phi^X(i, j)$ to be the maximum negative log probability among alignments of $\sigma_{\leq i}^1$ and $\sigma_{\leq j}^2$ such that the last character in the corresponding edit string is X . Equation (5.8) then gives us the following recursive formula(s):

$$\text{trop}(f_{\sigma^1, \sigma^2}) = \bigoplus_X \Phi^X(n, m), \tag{5.9}$$

where

$$\begin{aligned} \Phi^I(i, j) &= w_{-, \sigma_j^2} \odot \bigoplus_X (\Phi^X(i, j-1) \odot w'_{X, I}), \\ \Phi^D(i, j) &= w_{\sigma_i^1, -} \odot \bigoplus_X (\Phi^X(i-1, j) \odot w'_{X, D}), \\ \Phi^H(i, j) &= w_{\sigma_i^1, \sigma_j^2} \odot \bigoplus_X (\Phi^X(i-1, j-1) \odot w'_{X, H}), \end{aligned} \tag{5.10}$$

and

$$\begin{aligned} \Phi^X(0, 0) &= 0 \text{ for all } X, \\ \Phi^X(0, j) &= 0 \text{ for all } X \neq I, \\ \Phi^X(i, 0) &= 0 \text{ for all } X \neq D, \\ \Phi^I(0, j) &= w_{-, \sigma_1^2} \odot \bigodot_{k=2}^j (w'_{I, I} \odot w_{-, \sigma_k^2}), \\ \Phi^D(i, 0) &= w_{\sigma_1^1, -} \odot \bigodot_{k=2}^i (w'_{D, D} \odot w_{\sigma_k^1, -}). \end{aligned}$$

The running time of the Needleman–Wunsch algorithm is $O(nm)$ as we perform a constant number of \oplus and \odot operations for each pair of indices (i, j) .

5.2 The polytope propagation algorithm

In this section we will describe parametric *maximum a posteriori probability* estimation for probabilistic models. Our goal is to explain how parametric MAP estimation is related to linear programming and polyhedral geometry. The parametric MAP estimation problem comes in two different versions. First there is the local one: given a particular choice of parameters determine the set of all parameters which have the same MAP estimate. This version is an important problem because it can be used to decide how sensitive the MAP estimate is to perturbations in the parameters. The global version of parametric MAP estimation problem asks for a partition of the space of parameters such that any two parameters lie in the same part if and only if they yield the same MAP estimate. We will show that for arbitrary statistical models, the local problem is solved by computing a certain polyhedral cone (the normal cone at a vertex of the Newton polytope) and the global problem is solved by computing a certain polyhedral fan (the normal fan of the Newton polytope). If the underlying statistical model has a sum-product decomposition, there

is a natural extension of the tropical sum-product algorithm which replaces numbers with polyhedra and solves the parametric MAP estimation problem.

We will now show how to perform the tropical sum-product algorithm in a general fashion, finding an explanation for all possible sets of parameters. Let us consider the polynomial

$$f(p) = \sum_{i=1}^d p_1^{e_{i1}} \cdots p_k^{e_{ik}}$$

and suppose that f is the density function associated to a statistical model where $p = (p_1, \dots, p_k)$ is the vector of parameters, and each possible sequence of hidden states corresponds to some monomial of f . See (5.3) as an example. We maintain this assumption throughout the rest of this chapter.

For a fixed value of p , finding an explanation is equivalent to finding the index j of the monomial of f with maximum value

$$j = \operatorname{argmax}_i \{p_1^{e_{i1}} \cdots p_k^{e_{ik}}\}.$$

Letting $w_i = -\log p_i$, this amounts to finding the index j of the monomial of f which minimizes the linear expression $e_j \cdot w = \sum_{i=1}^k w_i e_{j,i}$. We observe that e_j can be an explanation for some choice of parameters if and only if the point $P_j = (e_{j1}, \dots, e_{jk})$ is on the convex hull of the set $\{(e_{i1}, \dots, e_{ik}) : i \in [d]\}$, i.e., it is a vertex of the Newton polytope of f , $\operatorname{NP}(f)$.

The optimization problem of finding an explanation for fixed parameters w can therefore be interpreted geometrically as a linear programming problem on the Newton polytope $\operatorname{NP}(f)$. In the notation of Section 2.3, the optimization problem described above means finding $(e_{j,1}, e_{j,2}, \dots, e_{j,k}) \in \operatorname{face}_w(\operatorname{NP}(f))$. Conversely, the parametric version of this problem asks for the set of parameter vectors w for which a vertex P_j gives the explanation. In Section 2.3 it is shown that this is the cone in the normal fan of the polytope $\operatorname{NP}(f)$ which corresponds to the vertex P_j , namely $\mathcal{N}_{\operatorname{NP}(f)}(P_j)$. Constructing the normal fan $\mathcal{N}_{\operatorname{NP}(f)}$ therefore amounts to partitioning the parameter space into regions such that the explanation(s) for all sets of parameters in a given region is given by the polytope vertex(face) associated to that region. We can obtain $\operatorname{NP}(f)$ and $\mathcal{N}_{\operatorname{NP}(f)}$ through the *polytope propagation algorithm*, which is the polytope algebra version of the sum-product decomposition. We refer the reader to Section 2.3 for details on Newton polytopes, normal fans and the polytope algebra.

For example, solving the parametric version of (5.4) for hidden Markov models or (5.8) for sequence alignment amounts to finding the normal fan of the Newton polytopes $\operatorname{NP}(p_\tau)$ and $\operatorname{NP}(f_{\sigma^1, \sigma^2})$. As can easily be observed, in both examples our polynomials will have an exponential number of monomials. It is thus not feasible to compute the Newton polytope by first computing the polynomial explicitly. We will therefore make use of the recursive representations given by (5.6) and (5.10). Theorem 2.26 immediately gives us a recursive representation of the needed Newton polytopes: simply translate (5.6) and (5.10) into the polytope algebra of Section 2.3.

For the hidden Markov model we obtain the following:

$$\begin{aligned} \text{NP}(p_\tau) &= \bigoplus_{\sigma_n} (\text{NP}(\theta'_{\sigma_n \tau_n}) \odot \bigoplus_{\sigma_{n-1}} (\text{NP}(\theta'_{\sigma_{n-1} \tau_{n-1}} \theta_{\sigma_{n-1} \sigma_n}) \\ &\quad \odot \dots \odot \bigoplus_{\sigma_1} (\text{NP}(\theta'_{\sigma_1 \tau_1} \theta_{\sigma_1 \sigma_2})) \dots)). \end{aligned}$$

In the sequence alignment example, take $\mathcal{P}^I(i, j)$ to be the Newton polytope of the sum of the scores of all alignments of the two partial sequences $\sigma_{\leq i}^1$ and $\sigma_{\leq j}^2$ which end with an insertion. This corresponds to the sum of the weights of all paths from the origin to the insertion vertex of the $K_{3,3}$ corresponding to position (i, j) in the alignment graph of Figure 2.2. Define $\mathcal{P}^D(i, j)$ and $\mathcal{P}^H(i, j)$ similarly and (5.10) gives us

$$\text{NP}(f_{\sigma^1, \sigma^2}) = \bigoplus_X \mathcal{P}^X(n, m), \quad (5.11)$$

where

$$\begin{aligned} \mathcal{P}^I(i, j) &= \text{NP}(\theta_{-, \sigma_j^2}) \odot \bigoplus_X (\mathcal{P}^X(i, j-1) \odot \text{NP}(\theta'_{X, I})), \\ \mathcal{P}^D(i, j) &= \text{NP}(\theta_{\sigma_i^1, -}) \odot \bigoplus_X (\mathcal{P}^X(i-1, j) \odot \text{NP}(\theta'_{X, D})), \\ \mathcal{P}^H(i, j) &= \text{NP}(\theta_{\sigma_i^1, \sigma_j^2}) \odot \bigoplus_X (\mathcal{P}^X(i-1, j-1) \odot \text{NP}(\theta'_{X, H})), \end{aligned}$$

and

$$\begin{aligned} \mathcal{P}^X(0, 0) &= \{(0, \dots, 0)\} \text{ for all } X, \\ \mathcal{P}^X(0, j) &= \{(0, \dots, 0)\} \text{ for all } X \neq I, \\ \mathcal{P}^X(i, 0) &= \{(0, \dots, 0)\} \text{ for all } X \neq D, \\ \mathcal{P}^I(0, j) &= \text{NP}(\theta_{-, \sigma_1^2} \prod_{k=2}^j (\theta'_{I, I} \theta_{-, \sigma_k^2})), \\ \mathcal{P}^D(i, 0) &= \text{NP}(\theta_{\sigma_1^1, -} \prod_{k=2}^i (\theta'_{D, D} \theta_{\sigma_k^1, -})). \end{aligned}$$

The above decompositions naturally yield straightforward algorithms for computing the Newton polytopes $\text{NP}(p_\tau)$ and $\text{NP}(f_{\sigma^1, \sigma^2})$, and one can easily extend this method to any polynomial f with a sum-product decomposition. Once the polytope $\text{NP}(f)$ has been computed, the final step of our algorithm is to compute the normal fan $\mathcal{N}_{\text{NP}(f)}$.

To conclude our presentation of polytope propagation, we have to make two very important observations. First, we note that several sequences of hidden states may yield equivalent monomials. In general the coefficient of the monomial is equal to the number of different sequences of hidden states that map to it. The Newton polytope of f , and thus the polytope propagation algorithm, is completely oblivious to this aspect. One can however recover the

full set of hidden sequences corresponding to a given vertex by backtracking along the steps of the polytope propagation algorithm, a common technique in dynamic programming. In sequence alignment for example, finding the edit strings corresponding to an extremal vertex is the parametric version of finding all optimal paths in the alignment graph of Section 2.2.

The second observation is that our assumption that each set of hidden states maps to a monomial of f implicitly insures that the coefficient of that monomial is not zero: in other words that no cancellation occurs in the sum-product decomposition. In general, for polynomials representing marginal probabilities in a graphical model, all the coefficients in their sum-product representations are positive. This is clearly the case for our two running examples. In fact, as Newton polytopes are oblivious to coefficients, running polytope propagation instead of evaluating f and $\text{NP}(f)$ directly automatically insures that no vertices are omitted because of cancellation. Therefore, all sequences of values for the hidden variables which are optimal for some set of parameters are represented in the final polytope, which will contain $\text{NP}(f)$ if cancellation occurs.

5.2.1 A small alignment example

To illustrate our algorithm, we give below a small example of parametric sequence alignment under a highly simplified version of the scoring scheme of Section 2.2. The parameters of our model are a “reward” assigned to all matches and a “penalty” assigned to all mismatches and gaps. We disregard completely the scores assigned to transitions between hidden nodes. In the language of Section 2.2, this is equivalent to $w'_{X,Y} = 0$ for all $X, Y \in \{H, I, D\}$, $w_{a,a} = x$ for all $a \in \{\text{A, C, G, T}\}$ and $w_{a,b} = y$ for all $a, b \in \{\text{A, C, G, T, -}\}$, $a \neq b$. This model is commonly known as the *2-parameter model for sequence alignment*.

Notice that the absence of transition probabilities between hidden nodes eliminates the need for the triple recurrence present in the sum-product decomposition of the generalized scoring scheme. Letting $\Phi(i, j)$ denote the score of the best alignment of the sequences $\sigma^1_{\leq i}$ and $\sigma^2_{\leq j}$, we have:

$$\Phi(i, j) = (\Phi(i-1, j-1) \odot w_{\sigma^1_i, \sigma^2_j}) \oplus (\Phi(i-1, j) \odot y) \oplus (\Phi(i, j-1) \odot y). \quad (5.12)$$

In polytope algebra, let $\mathcal{P}(i, j) = \text{NP}(\Phi(i, j))$. The above relation becomes:

$$\mathcal{P}(i, j) = (\mathcal{P}(i-1, j-1) \odot \text{NP}(w_{\sigma^1_i, \sigma^2_j})) \oplus (\mathcal{P}(i-1, j) \odot (y)) \oplus (\mathcal{P}(i-1, j-1) \odot (y)),$$

where (x) denotes the Newton polytope with the single vertex $(1, 0)$ and (y) denotes the Newton polytope with the single vertex $(0, 1)$.

Figure 5.1 illustrates the polytope propagation algorithm for the alignment of the two sequences $\sigma^1 = \text{ATCG}$ and $\sigma^2 = \text{TCGG}$. At each cell in the matrix, the displayed polytope is the convex hull of the points given by alignments of the corresponding partial sequences. Each is obtained by taking the convex hull of the $(0, 1)$ translation of the polytope in the cell above (alignments ending with a deletion), the $(0, 1)$ translation of the polytope in the left cell (alignments

ending with an insertion), and the (1, 0) or (0, 1) translation of the polytope in the above-left cell (alignments ending with a match or mismatch, respectively).

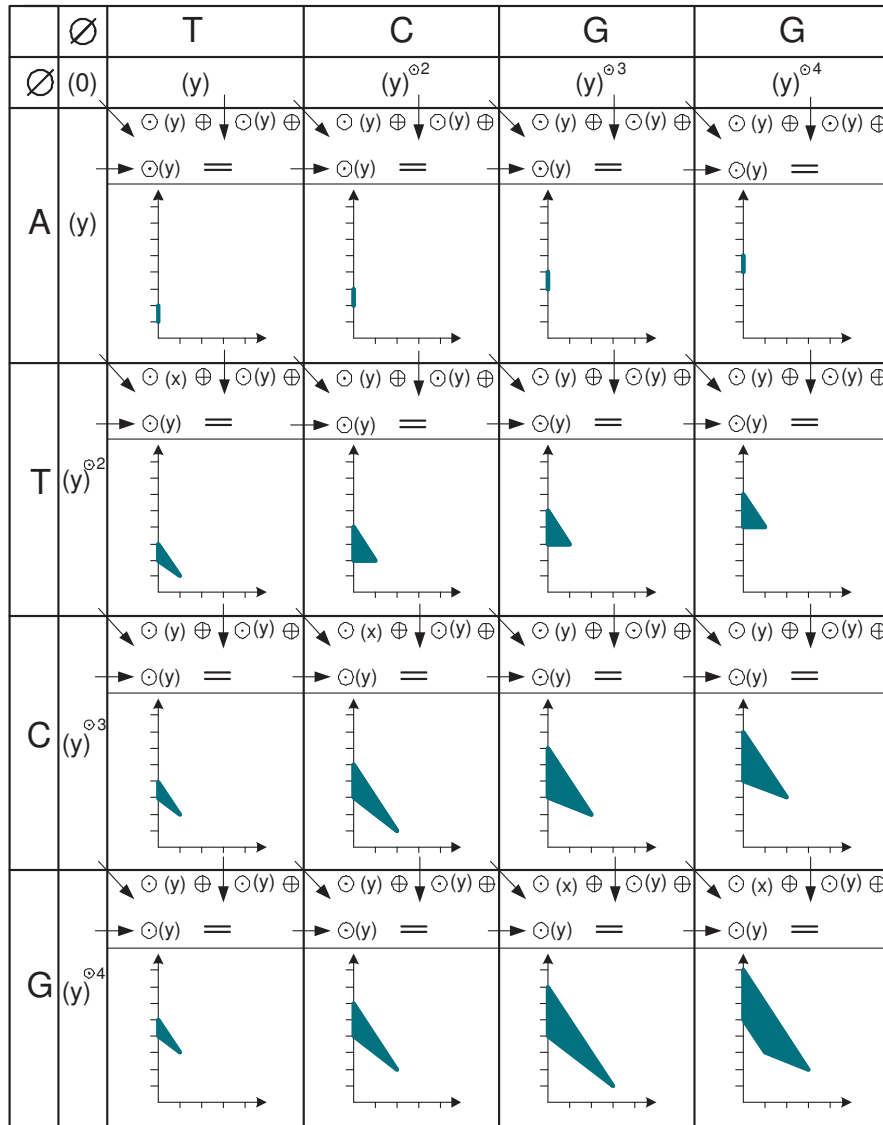


Fig. 5.1. Polytope propagation for sequence alignment.

5.3 Algorithm complexity

In this section we analyze the time complexity of the polytope propagation algorithm. Given a polynomial f together with a sum-product decomposition, we want to compute $\text{NP}(f)$ and $\mathcal{N}_{\text{NP}(f)}$. The questions we want to answer are:

- (i) How many polytope algebra operations do we have to perform?
- (ii) What is the individual time complexity of these operations?
- (iii) What is the complexity of computing the normal fan of $\mathcal{N}_{\text{NP}(f)}$?

Let us consider the first question. In the examples of the previous section, the only multiplicative (Minkowski sum) operations we needed to perform were multiplication by a single point, i.e., shifting a polytope by a given vector. For a D -dimensional polytope with N vertices, this takes $O(ND)$ time and is strongly dominated by the additive operations. In this section we will limit our analysis to models where the only multiplicative operations are the trivial ones, as this turns out to be the case with most acyclic graphical models.

In general, the number of polytope algebra operations we need to perform will be the product of the number of levels in the sum-product decomposition of f and the number of operations per level. This is exactly the time complexity of computing the explanation for a *given* set of parameters, using the sum-product decomposition. For instance, in the case of the hidden Markov model, the total number of polytope algebra operations will be $O(l^2n)$. In the sequence alignment example, at each step we compute three sums of exactly three polynomials, therefore the total number of operations is $O(nm)$.

In order to answer question (ii), we need to choose an efficient representation for our Newton polytopes. Section 2.3 provides us with two options: the V-representation and the H-representation. In general, the two are roughly equivalent in terms of computational versatility, due to the principle of duality. In the context of parametric inference, the V-representation is more natural, as we are able to prove upper bounds on the number of vertices of the Newton polytopes we encounter.

To facilitate our subsequent discussion, let $\nu_D(K)$ denote the computational complexity of finding the V-representation of the convex hull of K points in D dimensions. Also let N be the maximum number of vertices among all the polytopes encountered by the algorithm. It is clear that the additive polytope algebra operation will have a time complexity of at most $O(\nu_D(2N))$.

The next step is providing upper bounds for the number N . Since we are dealing with Newton polytopes of polynomials, all vertices will have integer coordinates. Now suppose that the degree of f is n . Then at every intermediate step of the algorithm, the degree of any variable will always be at most n . We can therefore assert that all polytopes encountered by the algorithm will lie inside the D -dimensional hypercube of side-length n . We will make use of the following theorem of [Andrews, 1963]:

Theorem 5.1 *For every fixed integer D there exists a constant C_D such that the number of vertices of any convex lattice polytope \mathcal{P} in \mathbb{R}^D is bounded above by $C_D \cdot \text{vol}(\mathcal{P})^{(D-1)/(D+1)}$.*

Unfortunately, Theorem 5.1 only applies to full dimensional polytopes: the polytope must not be contained in a lower-dimensional affine subspace of \mathbb{R}^D . As it turns out, this will almost always be the case. Now suppose that the final polytope we compute has dimension $d < D$. It is easy to see that the polytope algebra operations can only result in an increase of the dimension of the polytopes, i.e., $\dim(\mathcal{P} \oplus \mathcal{Q}) \geq \max\{\dim(\mathcal{P}), \dim(\mathcal{Q})\}$ and $\dim(\mathcal{P} \odot \mathcal{Q}) \geq \max\{\dim(\mathcal{P}), \dim(\mathcal{Q})\}$ for any two polytopes \mathcal{P} and \mathcal{Q} . Then all of

the intermediate polytopes will have dimension at most d . We call d the *true dimension* of the model or polynomial.

Let us illustrate. In the HMM example, each monomial $p_{\sigma,\tau}$ is a product of n variables θ'_{ij} and $n - 1$ variables θ_{ij} . Thus, for each point in $\text{NP}(p_\tau)$, the sum of the ll' coordinates corresponding to the θ'_{ij} variables is n and the sum of the l^2 coordinates corresponding to the θ_{ij} variables is $n - 1$. These two inherent constraints of the model mean that $d \leq D - 2$.

The following easy lemma will be needed in the derivation of our running time bounds.

Lemma 5.2 *Let \mathcal{S} be a d -dimensional linear subspace of \mathbb{R}^D . Then, among the set of D coordinate axes of \mathbb{R}^D , there exists a subset $\{i_1, i_2, \dots, i_d\}$ such that the projection $\phi : \mathcal{S} \rightarrow \mathbb{R}^d$ given by $\phi((x_1, x_2, \dots, x_D)) = (x_{i_1}, x_{i_2}, \dots, x_{i_d})$ is injective.*

Proof Let $v^1, \dots, v^d \in \mathbb{R}^D$ be a basis for the subspace \mathcal{S} . Let A be the $D \times d$ matrix with columns v^1, \dots, v^d . Then the rank of the matrix A is exactly d , the dimension of \mathcal{S} . Now suppose that for any choice of indices $\{i_1, i_2, \dots, i_d\}$ the projection $\phi((x_1, x_2, \dots, x_D)) = (x_{i_1}, x_{i_2}, \dots, x_{i_d})$ is not injective on \mathcal{S} . This means that the $d \times d$ minor of A given by the rows indexed by the i_j 's has rank strictly less than d . Since the choice of rows was arbitrary, this contradicts the fact that the rank of A is d . \square

Let \mathcal{S} be the d -dimensional affine span of a polytope \mathcal{P} . By Lemma 5.2, there is a set of d coordinate axes such that the projection ϕ of the linear subspace \mathcal{S} onto the space determined by those d axes is injective. Then $\phi(\mathcal{P})$ is also a d -dimensional polytope whose vertices have integer coordinates and are in 1-1 correspondence with the vertices of \mathcal{P} . Moreover, $\phi(\mathcal{P})$ lies inside a d -dimensional hypercube with edge length n (all the exponents are at most n). Therefore the volume of $\phi(\mathcal{P})$ is at most n^d . Applying Theorem 5.1, we infer that \mathcal{P} has at most $N = C_d \cdot n^{d(d-1)/(d+1)}$ vertices. Since all intermediate polytopes have dimension less than or equal to that of $\text{NP}(f)$, the above upper bound on the number of vertices will hold for all intermediate polytopes if we let d be the true dimension of our model. We obtain the following theorem:

Theorem 5.3 *Let f be a polynomial of degree n in D variables. If the dimension of $\text{NP}(f)$ is d , then all the polytopes computed by the polytope propagation algorithm will have at most $C_d \cdot n^{d(d-1)/(d+1)}$ vertices.*

We have not yet elucidated the mystery surrounding the function ν_D . It turns out that for $D = 2$ and $D = 3$, things are much simpler than in higher dimensions. The dominant operation in our algorithm is that of taking the convex hull of the union of two convex polytopes. For $D = 2$, this can be done in $O(N)$ time if the input polytopes have a total of N vertices, although computing the extremal vertices of a general set of N points takes $O(N \log N)$ time. For $D = 3$, computing the union of two disjoint convex polytopes with

N vertices takes $O(N)$ time, but when the polytopes are not disjoint it is asymptotically no easier than computing the convex hull of the union of their sets of vertices. This can be done in $O(N \log N)$ time by the algorithm of Preparata and Hong. The results for both 2 and 3 dimensions can be found in [Preparata and Shamos, 1985].

Now consider the case $D \geq 4$. A point $v_j = (x_{j1}, \dots, x_{jD}) \in \mathbb{R}^D$ is a vertex of the convex hull of the set $\mathcal{A} = \{v_1, \dots, v_N\}$ if and only if there is a hyperplane of \mathbb{R}^D separating v_j from the rest of the points in \mathcal{A} . In turn, the existence of such a hyperplane is equivalent to the feasibility of the following linear program:

$$\begin{aligned} &\text{Maximize } \epsilon \text{ subject to} \\ &\lambda_0, \dots, \lambda_D \in \mathbb{R}, \epsilon \geq 0, \\ &\sum_{k=1}^N \lambda_k x_{ik} \leq \lambda_0 - \epsilon \text{ for all } i \neq j \\ &\sum_{k=1}^N \lambda_k x_{jk} \geq \lambda_0 + \epsilon. \end{aligned} \tag{5.13}$$

We can therefore solve the problem of computing the extremal points of a set of N points in \mathbb{R}^D by solving N linear programs, each with D variables and N constraints. The time complexity of linear programming however is no simple matter. It is often the case that the algorithms which are optimal in some theoretical sense do not perform well in practice, whereas algorithms with no theoretical guarantees are very practical. See [Megiddo, 1984] and [Grötschel *et al.*, 1993] for more details.

If one regards the dimension D as fixed, solving a linear program with N constraints can be done $O(N)$ time by using the algorithms of [Megiddo, 1984]. However, the constant of proportionality is $O(2^{D \log D})$, which is exponential in D . On the other hand, Khachiyan's algorithm [Khachiyan, 1980] solves a linear program in time polynomial in the length of the bit representation of the program parameters. For our purposes, this implies a strongly polynomial algorithm, since all the points of our polytopes have integer coordinates of size at most n , therefore the linear programs will have integer parameters of size linear in n . The length of the representation of the linear programs will therefore be polynomial in n and d .

Theorem 5.4 *Let f be a polynomial of degree n in D variables, and suppose that f has a sum-product decomposition into k steps with at most l additions and l multiplications by a monomial per step. Let $d = \dim(\text{NP}(f))$ and let $N = C_d n^{d(d-1)/(d+1)}$. Then for $D = 2$ and $D = 3$, the V -representation of $\text{NP}(f)$ can be computed in time $O(klN)$ and $O(klN \log N)$, respectively. For $D \geq 4$, the V -representation of $\text{NP}(f)$ can be computed in time $O(kl\nu_D(2N))$, where $\nu_D(2N)$ is either $O(2^{O(D \log D)} N^2)$ or polynomial in both N and D .*

It is worth mentioning that in practice the well-known simplex method is often the fastest way to solve linear programs. The method relies on heuristic ways of moving along the boundary of a convex polytope called *pivoting strategies*. Although pivoting strategies which run in polynomial time have been found for special classes of linear programs, no such strategy is known for the general instance of linear programming [Megiddo, 1984].

Finally, we need to address our third question, the complexity of computing the normal fan of $\text{NP}(f)$. Note that if one is only interested in the set of extremal vertices of $\text{NP}(f)$, together with a single parameter vector for which each such vertex is optimal, then the V-representation of $\text{NP}(f)$ suffices and the computation of the normal fan is not needed. Linear programming provides us with a certificate for each vertex, i.e. a direction in which that vertex is optimal. If one is interested in the full set of parameter vectors associated to each vertex, then one needs to compute the normal fan $\mathcal{N}_{\text{NP}(f)}$. This requires the computation of the full convex hull of the polytope $\text{NP}(f)$ and its running time is in fact dominated by this computation. For $D \leq 3$, maintaining a full description of the convex hulls of our polytopes is no harder than maintaining the V-representation, however for fixed $D > 3$, an asymptotically optimal algorithm for computing the convex hull of a D -dimensional polytope with N vertices is given in [Chazelle, 1993] and has a time complexity of $O(N^{\lfloor D/2 \rfloor})$. Unfortunately, the constant of proportionality is again exponential in D .

Theorem 5.5 *Let f be a polynomial of degree n in D variables, and suppose that f has a sum-product decomposition into k steps with at most l additions and l multiplications by a monomial per step. Let $d = \dim(\text{NP}(f))$ and let $N = C_d n^{d(d-1)/(d+1)}$. Then for $D = 2$ and $D = 3$, $\mathcal{N}_{\text{NP}(f)}$ can be computed in time $O(klN)$ and $O(klN \log N)$, respectively. For $D \geq 4$, $\mathcal{N}_{\text{NP}(f)}$ can be computed in time $O(klN^2 + N^{\lfloor D/2 \rfloor})$, where the constant of proportionality is exponential in D .*

5.4 Specialization of parameters

5.4.1 Polytope propagation with specialized parameters

In this section we present a variation of the polytope propagation algorithm in which we may specialize the values of some of the parameters. Let us return to the generic example

$$f(p) = \sum_{j=1}^n p_1^{e_{j1}} \cdots p_k^{e_{jk}}.$$

Now assume that we assign the values $p_i = a_i$ for $h < i \leq k$. Our polynomial becomes

$$f_a(p) = \sum_{j=1}^n p_1^{e_{1j}} \cdots p_h^{e_{hj}} a_{h+1}^{e_{(h+1)j}} \cdots a_k^{e_{kj}},$$

which we can write as

$$f_a(p) = \sum_{j=1}^n p_1^{e_{1j}} \cdots p_h^{e_{hj}} \exp(e_{(h+1)j} \ln a_{h+1} + \cdots + e_{kj} \ln a_k). \quad (5.14)$$

We can treat the number e as a general free parameter and this new representation of f_a as a polynomial with only $h + 1$ free parameters, with the only generalization that the exponent of the last parameter need not be an integer, but an integer combination of the logarithms of the specialized parameters.

Now suppose that the polynomial f has a sum-product decomposition. It is clear that such a decomposition automatically translates into a decomposition for f_a by setting p_i^x to $e^{x \cdot \ln p_i}$. Furthermore, a monomial $p_1^{e_{j1}} \cdots p_k^{e_{jk}}$ gives an explanation for some parameter specialization $p = b$ such that $b_i = a_i$ for $i > h$ if and only if $b \cdot e_j = \max_i \{b \cdot e_i\}$, so only if the corresponding vertex $(e_1, \dots, e_h, \sum_{i=h+1}^k e_i \ln a_i)$ is on the Newton polytope $\text{NP}(f_a)$ of f_a . We have reduced the problem to that of computing $\text{NP}(f_a)$ given the sum-product decomposition of f_a induced by that of f .

Finally, we have to give an association of each set of parameters (e_1, \dots, e_h) with a vertex of $\text{NP}(f_a)$. We can do this in two ways, both of which have comparable time complexity. The first method involves computing the normal fan of $\text{NP}(f_a)$, just as before. This gives a decomposition of the space into cones, each of which corresponds to a vertex of $\text{NP}(f_a)$. For all vectors of parameters with negative logarithms lying inside a certain cone, the corresponding explanation vertex is the one associated with that cone. However, the last parameter in the expression of f_a is the number e , therefore the only relevant part of the parameter hyperspace \mathbb{R}^{h+1} is the hyperplane given by $p_{h+1} = e$, so $-\ln p_{h+1} = -1$. The decomposition G_{f_a} of this hyperplane induced by the normal fan of $\text{NP}(f_a)$ in \mathbb{R}^{h+1} is what we are interested in. Every region in this decomposition is associated with a unique vertex on the *upper half* of $\text{NP}(f_a)$ (with respect to the $(h + 1)$ th coordinate).

Alternatively we can compute the decomposition G_{f_a} directly. First we project the upper side of the polytope $\text{NP}(f_a)$ onto the first h coordinates. This gives a regular subdivision R_{f_a} of an h -dimensional polytope. The reason for projecting the upper side alone is that we are looking for minima of linear functionals given by the negative logarithms of the parameter vectors. However, the last parameter is e , so the corresponding linear coefficient is -1 . Taking the real line in \mathbb{R}^{h+1} corresponding to a fixed set of values for the first h coordinates, we can see that the point on the intersection of this line with the polytope $\text{NP}(f_a)$ which minimizes the linear functional is the one with the highest $(h + 1)$ th coordinate. Thus when projecting on the first h coordinates we will only be interested in the upper half of $\text{NP}(f_a)$.

In order to partition the hyperplane \mathbb{R}^h into regions corresponding to vertices of R_{f_a} , we need to identify the dividing hyperplanes in \mathbb{R}^h . Each such hyperplane is given by the intersection of the hyperplanes in the normal fan of $\text{NP}(f_a)$ with the h -dimensional space given by setting the last coordinate in \mathbb{R}^{h+1} to -1 . Therefore, each dividing hyperplane corresponds uniquely to

an edge (v_i, v_j) in R_{f_a} , and is given by the set of solutions (x_1, \dots, x_h) to the following linear equation:

$$\begin{aligned} x_1 e_{i1} + \dots + x_h e_{ih} - [e_{(h+1)i} \ln a_{h+1} + \dots + e_{ki} \ln a_k] = \\ x_1 e_{j1} + \dots + x_h e_{jh} - [e_{(h+1)j} \ln a_{h+1} + \dots + e_{kj} \ln a_k]. \end{aligned} \quad (5.15)$$

The subdivision of \mathbb{R}^h induced by these hyperplanes will be geometrically dual to R_{f_a} , with each region uniquely associated to a vertex of R_{f_a} , so of $\text{NP}(f_a)$. This is the object which we are interested in, as it gives a unique monomial of f for every set of values for the first h parameters, given the specialization of the last $k - h$ parameters.

5.4.2 Complexity of polytope propagation with parameter specialization

Let us now compute the running time of the algorithm described above. Much of the discussion in the previous section will carry through and we will only stress the main differences. As before, the key operation is taking convex hulls of unions of polytopes. Let N be the maximum number of vertices among all the intermediate polytopes we encounter. We again define the function $\nu_{h+1}(N)$ to be the complexity of finding the vertices of the convex hull of a set of N points in \mathbb{R}^{h+1} . Note that in the case of parameter specialization, the last coordinate of the vertices of our polytopes is not necessarily an integer.

This last observation implies that we will not be able to use Khachiyan's algorithm for linear programming to get a strongly polynomial algorithm for finding the extremal points. However, in practice this will not be a drawback, as one is nevertheless forced to settle for a certain floating point precision. Assuming that the values a_i we assign to the parameters p_i for $h < i \leq k$ are bounded, we may still assume that the binary representation of the linear programs needed to find the extremal points is polynomial in N . On the other hand, Megiddo's algorithm for linear programming is strongly polynomial in N for fixed h . It will run in time $O(2^{O(h \log h)} N)$. Of course, for the special case when our algorithms run in 2 or 3 dimensions, so $h = 1$ or $h = 2$, the analysis and running time bounds of the previous section still apply. We do not include them here for the sake of brevity.

Finally, what is an upper bound on N ? Our polytopes have vertices with coordinate vectors $(e_1, \dots, e_h, \sum_{i=h+1}^k e_i \ln a_i)$. By projecting on the first h coordinates, we see that each vertex must project onto a lattice point $(e_1, \dots, e_h) \in \mathbb{Z}_{\geq 0}^h$ such that $e_1, \dots, e_h \leq n$, where n is the degree of f . There are n^h such points. Moreover, at most two vertices of the polytope can project to the same point in \mathbb{R}^h . Therefore, $N \leq 2n^h$ and we have the following theorem.

Theorem 5.6 *Let f be a polynomial of degree n in D variables, and suppose that f has a sum-product decomposition into k steps with at most l additions and l multiplications by a monomial per step. Also suppose that all but h of*

f 's variables are specialized. Then the running time required to compute a V -representation of the Newton polytope of f with all but h parameters specialized is $O(kl\nu_{h+1}(N))$, where $N = 2n^h$ and $\nu_{h+1}(N) = O(2^{O(h \log h)}N^2)$.

For the last part of our task, computing the normal fan $\mathcal{N}_{\text{NP}(f_a)}$ and the regular subdivision it induces on the hyperplane $-\ln e_{h+1} = -1$, we remark that the running time is dominated by the computation of the full description of the convex hull of $\text{NP}(f_a)$. Again, if we consider h fixed, Chazelle's algorithm solves this in $O(N^{\lfloor (h+1)/2 \rfloor})$ time, where the constant of proportionality is exponential in h .

Theorem 5.7 *Let f be a polynomial of degree n in D variables, and suppose that f has a sum-product decomposition into k steps with at most l additions and l multiplications by a monomial per step. Also suppose that all but h of f 's variables are specialized. Then the running time required to compute all extremal vertices of $\text{NP}(f_a)$, together with their associated sets of parameter vectors, is $O(kl\nu_{h+1}(N) + N^{\lfloor (h+1)/2 \rfloor})$, where $N = 2n^h$, $\nu_{h+1}(N) = O(N^2)$ and all constants of proportionality are exponentials in h .*

We therefore observe that if one disregards the preprocessing step of retrieving a sum-product decomposition of f_a from a decomposition of f , the running time of our algorithm will not depend on the total number of parameters, but only on the number of unspecialized parameters. Also, the complexity of the pre-processing step is only linear in the size of the decomposition of f . This may prove a very useful feature when one is interested in the dependence of explanations on only a small subset of the parameters.

Conclusions: In this chapter we have presented the polytope propagation algorithm as a general method for performing parametric inference in graphical models. For models with a sum-product decomposition, the running time of polytope propagation is the product between the size of the decomposition and the time required to perform a convex hull computation. If the number of parameters D is fixed, this is polynomial in the size of the graphical model. However, as D increases the asymptotic running time is exponential in D . For large numbers of parameters, we present a variation of the algorithm in which one allows only h of the parameters to vary. This variation is computationally equivalent to running the initial algorithm on a model of the same size with only $h + 1$ parameters.