Dynamic Programming and the Graphical Representation of Error-Correcting Codes

Stuart Geman, *Member, IEEE,* and Kevin Kochanek

***Abstract*—Graphical representations of codes facilitate the design of computationally efficient decoding algorithms. This is an example of a general connection between dependency graphs, as arise in the representations of Markov random fields, and the dynamic programming principle. We concentrate on two computational tasks: finding the maximum-likelihood codeword and finding its posterior probability, given a signal received through a noisy channel. These two computations lend themselves to a particularly elegant version of dynamic programming, whereby the decoding complexity is particularly transparent. We explore some codes and some graphical representations designed specifically to facilitate computation. We further explore a coarse-to-fine version of dynamic programming that can produce an exact maximum-likelihood decoding many orders of magnitude faster than ordinary dynamic programming.**

***Index Terms*—Dynamic programming, graphical models, maximum-likelihood decoding, soft decoding.**

I. Introduction

By now, nearly everyone in the coding community knows about the tight relationship between the dependency structure on a set of random variables and the costs of computing certain functionals on their joint distribution. One way to deduce this relationship is to generalize, more-or-less straightforwardly, the principle of dynamic programming (Bellman [1]), but it can be arrived at from many other directions as well. In fact, an amazing history of discovery and rediscovery emerges from a diverse range of applications. Indeed, in one way or another, the "forward–backward" algorithm [2], "peeling" [3], "parsing" [4], the Viterbi algorithm [5], the "sum–product" algorithm [6]–[9], "bucket elimination" [10], and "evidence propagation" [11]–[13], all rest on this very same dynamic-programming principle. Oftentimes there is a good reason for using one variation over another. With one approach, there may be a book-keeping scheme that avoids duplicating computations; with another, the computational flow may be particularly transparent or convenient. The best choice is typically dictated by the computational constraints and the functionals of interest.

Manuscript received January 1, 2000; revised September 5, 2000. This work was supported in part by the Army Research Office under Contract DAAH04-96-1-0445, the National Science Foundation under Grant DMS-9217655, and the Office of Naval Research under Contract N00014-97-0249.

S. Geman is with the Division of Applied Mathematics, Brown University, Providence, RI 02912 USA (e-mail: geman@dam.brown.edu).

K. Kochanek was with the Division of Applied Mathematics, Brown University, Providence, RI 02912 USA. He is now with Mathematical Technologies Inc., Providence, RI 02906 USA (e-mail: kochanek@mathtech.com).

Communicated by B. J. Frey, Guest Editor.

Publisher Item Identifier S 0018-9448(01)00727-1.

We will concentrate here on two particular kinds of computations: computation of the most likely configuration of a collection of random variables, and this configuration's associated probability. In the context of maximum-likelihood ("soft") decoding, we calculate these functionals under the *posterior* distribution—the conditional distribution on the transmitted codewords given the received signal. What codeword was most likely transmitted? What is the conditional probability that this maximum-likelihood codeword was in fact transmitted? In other words, we will compute the *maximum a posteriori* (MAP) estimator of the transmitted codeword and the (conditional) probability that the estimator is actually correct. MAP decoding gives the most likely codeword, and therefore minimizes the probability of a decoding error. If, alternatively, we seek to minimize the number of information-bit errors, then we would maximize the posterior marginal at each information bit. In general, these are not the same thing, and the better choice is clearly problem-dependent. MAP makes sense when the codeword represents a logical unit, perhaps a product name, a price, a category, or an English word. On the other hand, a stream of information bits with no *a priori* structure calls for minimizing the bit-error rate.

This paper is about using graphical representations of codes and probability models of channels to calculate the exact MAP decoding and its probability. The emphasis is on finding representations and algorithms that make these calculations computationally feasible. It should perhaps be pointed out that, although the connections between graphs, codes, and computation are currently the subjects of intense research, most of the effort is in a somewhat different direction. The emphasis, instead, is on iterative decoding algorithms that are generally not exact and whose convergence properties are generally not well understood, but often exhibit, nevertheless, spectacular performance. Many codes which would be impossible to decode exactly lend themselves effectively and efficiently to iterative decoding, in some cases even approaching channel capacity. Gallager's [14] low-density parity-check codes and associated iterative decoding algorithm are probably the first examples, but few if any recognized the power of Gallager's construction. Tanner [15] found clean graphical representations of Gallager and other related codes, and utilized these representations to address a host of design issues concerning computational complexity and minimum distance. Tanner's graphical representations helped to set the stage for the rediscovery of Gallager codes, and for the many extensions that have evolved over the past few years. (See also Bahl *et al.* [16] for an early paper anticipating, rather remarkably, the modern viewpoint.) Of course, the turbo codes of Berrou *et al.* [17] also spurred this line of re-

search, since these codes, as well, have natural graphical representations, come equipped with an iterative decoding algorithm, and appear to perform near optimally in certain regimes. Wiberg and collaborators (see [6] and [7]) picked up on Tanner graphs and turbo codes, and within a general treatment of codes on graphs made connections to soft decoding, general channel models (with memory), and iterative and noniterative decoding algorithms. MacKay and Neal (see MacKay [18] for a thorough discussion) developed some of these same themes, and, additionally, introduced related codes and decoding methods that appear to be among the current best performers. The connections and common themes among all of these approaches, as well as to a broader framework including Bayesian inference, Markov random fields, belief propagation, and the modern theory of expert systems, seem to have been first understood and most clearly formulated by Kschischang and Frey [19].

Our goal in this paper is twofold. We first present a brief tutorial on Markov random fields (MRFs), dependency graphs, and the noniterative computation of various functionals on marginal and posterior distributions. In the remainder of the paper, we present some new results on how to apply these computational strategies to facilitate the maximum-likelihood decoding of graphical codes. As we have said, our focus is somewhat different from the current trend, in that we concentrate on representations that allow exact computation, and on new methods for reducing computational complexity. But in light of the often good performance of iterative methods, it would be of great interest to systematically compare iterative and exact computations in a decoding problem that lends itself to both approaches. We have not yet made these comparisons.

Section II is about generic computational issues on graphs. There is an especially transparent connection between graphs that represent dependency structures and the efficient calculation of a most likely configuration and its associated probability. This is essentially the "bucket elimination" algorithm, and it is perhaps the most straightforward of the various generalizations of dynamic programming. There is no need to triangulate, no need to construct junction trees, and no need to worry about cycles. Conveniently, the computational cost is readily calculated once a site-visitation schedule has been established. Furthermore, it is often the case that the most efficient site-visitation schedule is immediately apparent from the general structure of the dependency graph.

In Section III, we consider the simplest possible case: graphs with linear structure. We review their connection to convolutional codes and revisit Viterbi decoding from the MRF viewpoint. We point out that the (posterior) probability that the Viterbi-decoded signal is correct can be calculated as easily as the decoding itself, and we discuss extensions to channels with Markov memory, designed to model burst noise [20], [7]. In Section IV, we generalize to the case of tree-structured graphs and make a connection to production systems and context-free grammars. We reformulate Forney's squaring construction [21] to obtain a grammatical representation of the Reed–Muller (RM) and other related codes. Moreover, we discuss the computational complexity of soft decoding or of evaluating posterior probabilities for both memoryless and Markov communications channels. Finally, in Section V, we introduce two (not necessarily exclusive) methods for reducing the computational demands of maximum-likelihood decoding. The first is a "thinning" algorithm which controls computational costs by reducing information density. The second, coarse-to-fine dynamic programming [22], is a kind of multiscale version of dynamic programming that produces a provably optimal configuration, often with greatly reduced computation. We present an exact coarse-to-fine algorithm for some "context-free" codes, including the RM codes, and demonstrate thousandsfold improvements in decoding efficiency.

II. Dependency Graphs and Computing

Given a collection of random variables $X_1, \ldots, X_n$ and a probability distribution

$$P(x_1, \ldots, x_n) = \text{Prob}\{X_1 = x_1 \ldots X_n = x_n\}$$

how difficult is it to compute things like the marginal distribution on X_1, or the most likely configuration $x_1, \ldots, x_n$? More than anything else, the dependency relationships among the random variables dictate the complexity of computing these and other functionals of P. Dependency relationships can be conveniently represented with a graphical structure, from which the complexity of various computations can be more or less "read off." These graphical representations, and their connection to computing, are the foundation of modern expert systems [11]–[13] as well as of speech-recognition technologies [2], [23]. In fact, the connection is quite general, having emerged and re-emerged in these and many other application areas, such as genetics [3], coding theory [5], [17], computational linguistics [4], and image analysis [24].

This connection between graphs and computing is fundamental to the modern treatment of soft decoding, and we begin here with a brief tutorial. (See also Frey [25] for an excellent introduction to some of the same material.) Our approach to graphs is through Markov random fields, and our computational focus is on computing most likely configurations and their corresponding probabilities.

A. Markov Random Fields and Their Gibbs Representations

For the purpose of constructing dependency graphs, it is convenient to index random variables by a general finite index set S, rather than the more traditional set $\{1, 2, \ldots, n\}$. If state spaces are finite, then $X = \{X_s\}_{s \in S}$ is a finite vector of random variables with finite range. In most applications, different components have different state spaces, but merely for the sake of cleaner notation, we will assume a common (and finite) range $\mathcal{R}$. Thus, P is a probability on $\mathcal{R}^S$.

P is a *Markov random field* (MRF) with respect to a graph $\mathcal{G} = \{S, \mathcal{N}\}$ if P is strictly positive ($P(x) > 0, \forall x \in \mathcal{R}^S$) and if

$$P(x_s|_s x) = P(x_s|\{x_t\}_{t \in \mathcal{N}_s}) \tag{1}$$

for all x and all $s \in S$, where

- S indexes the nodes of $\mathcal{G}$;

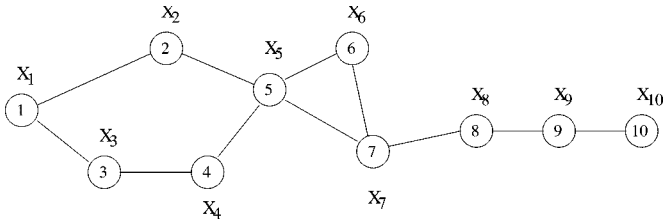

Fig. 1. Dependency graph for a simple Markov random field.

- $\mathcal{N} = \{\mathcal{N}_s\}_{s \in S}$ is the neighborhood structure of $\mathcal{G}$, meaning that $t \in \mathcal{N}_s$ if and only if $t \neq s$ and there is an edge connecting t and s in $\mathcal{G}$; and
- ${}_sx$ is shorthand for $\{x_t\}_{t \in S \setminus s}$.

The distribution on the random variable associated with any site $s \in S$, given the values of all other random variables, depends only on the values of the neighbors. This generalizes the familiar one-sided Markov property.

An example with $S = \{1, 2, \ldots, 10\}$ is in Fig. 1. The graph summarizes many (conditional) independence relationships. For instance,

$$P(x_7|x_1, x_2, x_3, x_4, x_5, x_6, x_8, x_9, x_{10}) = P(x_7|x_5, x_6, x_8)$$
$$P(x_1|x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = P(x_1|x_2, x_3).$$

P is *Gibbs* with respect to a graph $\mathcal{G} = \{S, \mathcal{N}\}$ if P can be represented as

$$P(x) = \prod_{c \in \mathcal{C}} F_c(x_c) \tag{2}$$

where

- $\mathcal{C}$ is the set of cliques in $\mathcal{G}$, i.e., the set of fully connected subsets of S (including singletons);
- $x = \{x_s\}_{s \in S}$, $x_c = \{x_s\}_{s \in c}$; and
- each F_c is a positive function.

Such representations are certainly not unique: constants can be "shifted" (multiply one term, divide another) and F_c can be absorbed into $F_{c'}$ whenever $c \subseteq c'$.

The main tool for working with MRFs is the representation theorem of Hammersley and Clifford [26], [27]: P is MRF with respect to (wrt) $\mathcal{G}$ if and only if P is Gibbs wrt $\mathcal{G}$. One direction is easy: it is straightforward to verify that (2) has the required Markov property. But the other direction (MRF wrt $\mathcal{G} \Rightarrow$ Gibbs wrt $\mathcal{G}$) is not easy. It does, though, make easy the proof that MRFs wrt linear graphs are Markov processes (i.e., that (1) implies the more familiar "one-sided" Markov property).

Referring to Fig. 1, the cliques are the singletons, the pairs $\{\{1, 2\}, \{1, 3\}, \{3, 4\}, \{2, 5\}, \{4, 5\}, \{5, 6\}, \{5, 7\}, \{6, 7\}, \{7, 8\}, \{8, 9\}, \{9, 10\}\}$, and the triple $\{5, 6, 7\}$. P MRF wrt the graph in Fig. 1 means that P can be factored into terms each of which depends only on the components represented in one of these cliques. An analogous relationship appears in the study of Bayes nets, which use directed acyclic graphs (DAGs). If P "respects" a DAG (if P factors into conditional probabilities of individual daughter nodes given their parent nodes) then P is Markov wrt the corresponding undirected "moral" graph (turn arrows into edges and then connect all parents of each daughter).[1]

If our only interest is in graphical representations of Gibbs distributions, then strict positivity can be relaxed. In particular, even if we drop the condition $F_c(x_c) > 0$ in the definition of Gibbs distributions, we still have the Markov property (1) wrt $\mathcal{G}$, provided that we avoid conditioning on events with probability zero. The *computational* analysis of graphical models, as it turns out, relies only on the implication Gibbs $\Rightarrow$ MRF. Therefore, we shall proceed without the positivity constraint $(F_c > 0)$, which would otherwise be troublesome in some of our applications.

B. Marginal and Posterior Distributions

In just about every application of MRFs, including coding, we are interested in making inferences about a subset of the variables given observations of the remaining variables. What makes MRFs useful in these applications is the fortunate fact that the *conditional distribution* on the unobserved variables given the observed variables (i.e., the *posterior* distribution) is an MRF on the subgraph of $\mathcal{G}$ obtained by restricting to "unobserved" sites. The conditional distribution on $X_1, X_2, X_5, X_7, X_8, X_9, X_{10}$, given X_3, X_4, and X_6—see Fig. 1—is Markov wrt the subgraph at sites $\{1, 2, 5, 7, 8, 9, 10\}$, which in this case happens to be linear. Up to a constant, conditional distributions are just joint distributions with some of the variables fixed, so the statement about their dependency structure follows immediately from the Gibbs representation.

Hidden Markov models (speech recognition [23], Kalman filters [28], etc.), and hidden Markov random fields [29], have dependency graphs like those in Fig. 2, where we have labeled the observable variables of the model using y's, and the unobservable ones using x's, in order to distinguish them. In Fig. 2(a) and (b), the posterior distribution, $P(x|y)$, is Markov wrt the linear graph; in Fig. 2(c) it is Markov wrt the nearest neighbor lattice graph.

The goal is usually to make an inference about X, given Y. Therefore, it is significant that the graphical structure representing the distribution on X given Y is no more complicated than the original structure, since, as we shall see shortly, this structure determines computational complexity. In this regard, it is reassuring that models such as those in Fig. 2 form a very rich class: the set of *marginal distributions* on Y, obtainable from finite-state space hidden nearest neighbor Markov models is essentially everything (up to arbitrary approximation, see [29]). One way to understand this is to examine the graphical structure of the marginal distribution on Y. Consider the Gibbs representation: when the X variables are summed (integrated) out, new cliques are introduced. Two sites, s and t, in the Y subgraph will be connected under the marginal distribution on Y (after integrating out X) if s and t are already connected under the joint (X, Y) distribution, *or* if there exists a path, traveling strictly through the X variables, that connects s and t. So the marginal on Y, in the cases depicted in Fig. 2, will in general define a fully connected graph! (Just check that there is always a path

[1]Another variant is the Tanner graph (cf. [15], [25]), in which clique functions are represented explicitly as specially designated nodes in the dependency graph.

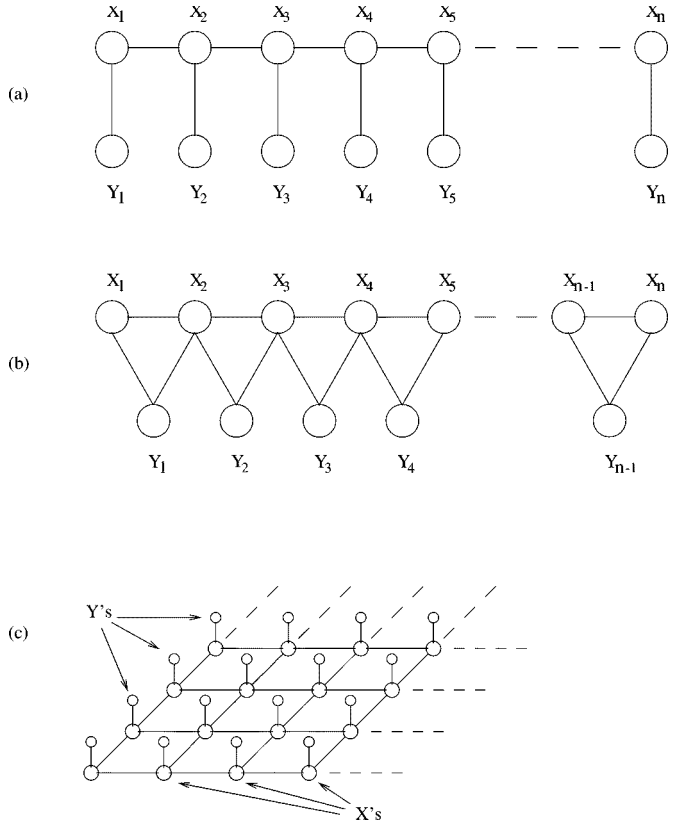

Fig. 2. (a) and (b) Hidden Markov models, and (c) a hidden Markov random field .

"through X" connecting any two components of Y.) This observation comes (easily—again!) from the Gibbs representation. (The condition for creating neighbors in a marginal distribution is necessary, but not quite sufficient—hence the disclaimer "in general." Consider, for example,

$$P(x, y_1, y_2) = P(x|y_1, y_2)P(y_1)P(y_2).$$

Then Y_1 and Y_2 are marginally independent

$$P(y_1, y_2) = P(y_1)P(y_2)$$

yet they are neighbors in the joint distribution *and* they are connected by a path through X.)

C. Computation

1) Most Likely Configurations: Computing a most likely sequence of words given an acoustic signal, a most likely image restoration given a corrupted picture, or a most likely codeword given a channel output, means computing a most likely configuration under a posterior distribution. In light of our remarks about the dependency structure of conditional distributions (Section II-B), it is evident that the generic problem is to maximize a probability distribution that is Gibbs relative to some given graph $\mathcal{G}$.

Consider again the simple example in Fig. 1. If, for instance, $\mathcal{R} = \{1, 2, \ldots, 20\}$ then there are 20^{10} possible configurations and exhaustive search for the most likely one is impractical. On the other hand, we could use a kind of (generalized) dynamic programming: choose first an ordering of the nodes in $\mathcal{G}$, say 10, 9, 8, 7, 6, 5, 4, 2, 3, 1. In general, P has the factorization

$$\begin{aligned} P(x) = &F_{1,2}(x_1, x_2)F_{1,3}(x_1, x_3)F_{2,5}(x_2, x_5)F_{3,4}(x_3, x_4) \\ &\times F_{4,5}(x_4, x_5)F_{5,6,7}(x_5, x_6, x_7)F_{7,8}(x_7, x_8) \\ &\times F_{8,9}(x_8, x_9)F_{9,10}(x_9, x_{10}) \end{aligned}$$

where, for convenience, we have absorbed sub-cliques into super-cliques. Following the site ordering, address x_{10} first: compute

$$\mathcal{R}_{10}(x_9) = \arg\max_{x\in\mathcal{R}} F_{9,10}(x_9, x)$$

and

$$C_{10}(x_9) = F_{9,10}(x_9, \mathcal{R}_{10}(x_9)).$$

Notice that x_9 "isolates" x_{10} from the other components, and $\mathcal{R}_{10}(v)$ is the value of x_{10} that participates in the most likely configuration, if that configuration involves $x_9 = v$. $C_{10}(v)$ is the corresponding contribution from terms involving x_{10}, evaluated at $x_9 = v$. Since x_9 is next on the list, and x_8 isolates x_9 and x_{10} from the other components, compute

$$\mathcal{R}_9(x_8) = \arg\max_{x\in\mathcal{R}} \{F_{8,9}(x_8, x)C_{10}(x)\}$$

and

$$C_9(x_8) = F_{8,9}(x_8, \mathcal{R}_9(x_8))C_{10}(\mathcal{R}_9(x_8))$$

to get, respectively, the value of x_9 participating in the optimal configuration (given x_8) as well as the contribution from terms involving x_9 and x_{10}. As for x_8

$$\mathcal{R}_8(x_7) = \arg\max_{x\in\mathcal{R}} \{F_{7,8}(x_7, x)C_9(x)\}$$

and

$$C_8(x_7) = F_{7,8}(x_7, \mathcal{R}_8(x_7))C_9(\mathcal{R}_8(x_7)).$$

So far this is standard dynamic programming, but x_7 calls for a slight generalization. Since it takes *both* x_5 and x_6 to isolate x_7, x_8, x_9, and x_{10} from the remaining variables, compute

$$\mathcal{R}_7(x_5, x_6) = \arg\max_{x\in\mathcal{R}} \{F_{5,6,7}(x_5, x_6, x)C_8(x)\}$$

and

$$C_7(x_5, x_6) = F_{5,6,7}(x_5, x_6, \mathcal{R}_7(x_5, x_6))C_8(\mathcal{R}_7(x_5, x_6)).$$

Proceeding in this way, always isolating what has been done from what has not been done, we compute

$$\begin{aligned} \mathcal{R}_6(x_5) &= \arg\max_{x\in\mathcal{R}} C_7(x_5, x) \\ C_6(x_5) &= C_7(x_5, \mathcal{R}_6(x_5)) \\ \mathcal{R}_5(x_2, x_4) &= \arg\max_{x\in\mathcal{R}} \{F_{2,5}(x_2, x)F_{4,5}(x_4, x)C_6(x)\} \\ C_5(x_2, x_4) &= F_{2,5}(x_2, \mathcal{R}_5(x_2, x_4))F_{4,5}(x_4, \mathcal{R}_5(x_2, x_4)) \\ &\quad \times C_6(\mathcal{R}_5(x_2, x_4)) \\ \mathcal{R}_4(x_2, x_3) &= \arg\max_{x\in\mathcal{R}} \{F_{3,4}(x_3, x)C_5(x_2, x)\} \\ C_4(x_2, x_3) &= F_{3,4}(x_3, \mathcal{R}_4(x_2, x_3))C_5(x_2, \mathcal{R}_4(x_2, x_3)) \\ \mathcal{R}_2(x_1, x_3) &= \arg\max_{x\in\mathcal{R}} \{F_{1,2}(x_1, x)C_4(x, x_3)\} \\ C_2(x_1, x_3) &= F_{1,2}(x_1, \mathcal{R}_2(x_1, x_3))C_4(\mathcal{R}_2(x_1, x_3), x_3) \\ \mathcal{R}_3(x_1) &= \arg\max_{x\in\mathcal{R}} \{F_{1,3}(x_1, x)C_2(x_1, x)\} \\ C_3(x_1) &= F_{1,3}(x_1, \mathcal{R}_3(x_1))C_2(x_1, \mathcal{R}_3(x_1)) \\ \mathcal{R}_1 &= \arg\max_{x\in\mathcal{R}} C_3(x). \end{aligned}$$

$\mathcal{R}_1$ is the value of x_1 that participates in the most likely configuration (with the corresponding cost $C_1 = C_3(\mathcal{R}_1)$). Evidently, then, $\mathcal{R}_3(\mathcal{R}_1)$ is the corresponding value of x_3, and $\mathcal{R}_2(\mathcal{R}_1, \mathcal{R}_3(\mathcal{R}_1))$ the corresponding value of x_2, and so on, back through the graph.

The most likely configuration x is thereby computed with many fewer operations than required in a systematic (brute-force) search. How many operations are needed? The worst of it is in the computation of $\mathcal{R}_7$, $\mathcal{R}_5$, $\mathcal{R}_4$, and $\mathcal{R}_2$, each of which involves a triple loop (e.g., with regard to $\mathcal{R}_7$: for every x_5 and every x_6 find the best x_7) and hence order $|\mathcal{R}|^3$ operations. Since there are 10 sites, and no site requires more operations than $O(|\mathcal{R}|^3)$ an upper bound on the computational cost is $O(10|\mathcal{R}|^3)$.

This procedure generalizes. For any graph $\mathcal{G}$ with n sites let $s_1, s_2, \ldots, s_n \in S$ be a "site visitation" schedule. How much work is involved in "visiting" a site? When visiting s_k, we need to compute the best x_{s_k} for each possible configuration on the set of sites *that isolate* $s_1, s_2, \ldots, s_k$ from the remaining sites. In other words, if N_k is the number of neighbors of $s_1, s_2, \ldots, s_k$ among $s_{k+1}, \ldots, s_n$ in the graph $\mathcal{G}$ (the "size of the boundary set of $s_1, s_2, \ldots, s_k$"), then there are $O(|\mathcal{R}|^{N_k+1})$ operations associated with the visit to k (N_k neighbors plus a loop through the states of x_{s_k}). Therefore, if $N_{\max} = \max\{N_1, N_2, \ldots, N_n\}$, then the maximum-likelihood configuration can be found in

$$O(n|\mathcal{R}|^{N_{\max}+1})$$

operations.

Actually, things can be better than this. Suppose $s_1, s_2, \ldots, s_k$ is not connected (in the graph structure of $\mathcal{G}$). The maximization on x_{s_k} is not necessarily dependent upon all of the bounding variables of the set $x_{s_1}, \ldots, x_{s_k}$. In fact, one need only fix the values of the variables at those sites that isolate the *particular connected component containing* s_k. Thus, $N_{\max}$ should be interpreted, more favorably, as the *largest boundary of a connected component* created by the site visitation schedule $s_1, \ldots, s_n$.

This makes a substantial difference. In Fig. 3, with visitation schedule 1, 3, 5, 7, 2, 4, 6, 8, 9, no connected set is generated with boundary size greater than one: $N_{\max} = 1$ and only $O(9|\mathcal{R}|^2)$ operations are needed. Of course order matters: any visitation schedule that starts with x_9 will immediately incur $O(|\mathcal{R}|^5)$ computations.

Remarks:

1) None of this would really work on a big problem, say with $n = 100$, even if $N_{\max}$ were small. The C functions represent probabilities, and when k is large, the probability of any configuration $x_{s_1}, x_{s_2}, \ldots, x_{s_k}$ is exponentially small and would generate an underflow. Therefore, in practice, we maximize the logarithm of P instead of P itself. Products (such as $F_{2,9}(x_2, x_9)C_1(x_2)$ above) are replaced by sums (such as $\log F_{2,9}(x_2, x_9) + \log C_1(x_2)$), but otherwise the procedure and the reasoning behind it are the same.

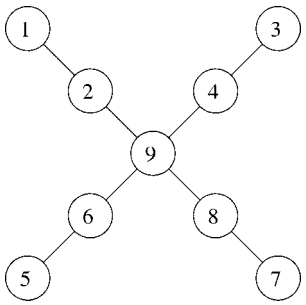

Fig. 3. Star-shaped dependency graph.

2) Sometimes, an $\arg\max$ will produce a tie. These can be decided arbitrarily, in which case one of possibly many maximum probability configurations will be found.
3) There is a more-or-less obvious modification that finds the k most likely configurations, for any $k \geq 1$.
4) Often an *optimal* ordering, in the sense of achieving the minimum $N_{\max}$, is transparent, more or less by inspection. But the problem in general, for arbitrary graphs, is NP-hard. See Arnborg *et al.* [30].

2) Marginal and Conditional Probabilities: Often, the object of interest is a marginal or conditional probability distribution on a subset of the variables. What is the probability of congestive heart failure in a 50-year-old male with swollen ankles and pneumonia? What is the probability that a particular decoding, for example, the maximum-likelihood decoding, is actually correct given the output of a noisy channel and given that codewords are (say) *a priori* equally likely? These conditional probabilities are quotients of marginal probabilities—probabilities on configurations of subsets of the variables. The coding example requires the marginal probability of the observed channel output, and this involves a summation of probabilities over all possible inputs; for a medical application, we may need to compute marginal probabilities on small subsets of variables, associated with diseases like congestive heart failure and pneumonia, attributes like age and sex, and signs and symptoms like swollen ankles.

The general problem is, therefore, to compute the probability of a configuration of states for a subset of variables, given a Gibbs distribution on an associated graph $\mathcal{G}$. There is again a dynamic programming principle, operating in much the same way as in the calculation of a most likely configuration. This is probably best illustrated by example.

Consider again Fig. 3, and suppose we wish to calculate the marginal distribution on (x_1, x_3). Then for each value of x_1 and x_3 we will need to sum out the other seven variables

$$\begin{aligned} P(x_1, x_3) &= \sum_{x_2, x_4, x_5, x_6, x_7, x_8, x_9 \in \mathcal{R}} F_{1,2}(x_1, x_2)F_{2,9}(x_2, x_9) \\ &\quad \times F_{3,4}(x_3, x_4)F_{4,9}(x_4, x_9)F_{5,6}(x_5, x_6)F_{6,9}(x_6, x_9) \\ &\quad \times F_{7,8}(x_7, x_8)F_{8,9}(x_8, x_9). \end{aligned}$$

The apparent computational cost is $|\mathcal{R}|^7$, but if we pay attention to the *order* of summation then this can be reduced to less than $7|\mathcal{R}|^2$. We again define a site visitation schedule, say 5, 7, 2, 4, 6, 8, 9, and t his a gain dictates the sequence of

calculations: define

$$T_5(x_6) = \sum_{x \in \mathcal{R}} F_{5,6}(x, x_6)$$
$$T_7(x_8) = \sum_{x \in \mathcal{R}} F_{7,8}(x, x_8)$$
$$T_2(x_1, x_9) = \sum_{x \in \mathcal{R}} F_{2,9}(x, x_9) F_{1,2}(x_1, x)$$
$$T_4(x_3, x_9) = \sum_{x \in \mathcal{R}} F_{4,9}(x, x_9) F_{3,4}(x_3, x)$$
$$T_6(x_9) = \sum_{x \in \mathcal{R}} F_{6,9}(x, x_9) T_5(x)$$
$$T_8(x_9) = \sum_{x \in \mathcal{R}} F_{8,9}(x, x_9) T_7(x)$$

and then finally

$$P(x_1, x_3) = T_9(x_1, x_3) = \sum_{x \in \mathcal{R}} T_8(x) T_6(x) T_4(x_3, x) T_2(x_1, x).$$

In essence, the two schemes, for maximizing and for summing, are the same. Pick a visitation schedule, fix the variables on the boundary of the connected set containing the current site, and then either maximize or sum. In either case, the number of elementary computations is no worse than

$$O(n|\mathcal{R}|^{N_{\max}+1}).$$

As we have said, conditional probabilities are quotients of marginal probabilities, so conditional probabilities are also amenable to dynamic programming. But this will not always work! At least not when there are a large number of "observed" variables—variables upon which we condition. The problem is again numerical: the *a priori* probability of any one configuration of the observable variables is exponentially small, but this very same probability is the denominator of the quotient representing the desired conditional probability. If, for instance, a block code transmits 1024 bits, then the *unconditioned* probability of receiving any particular 1024-bit word is hopelessly small—much too small to be computed with a summation scheme like the one recommended above. On the other hand, the *conditional* probability of, say, the *most likely* transmitted word, given the received word, will typically be order one. What we are after is a ratio of two very small numbers, and we need to use caution to avoid gross numerical error.

One way around this is to mix the computations of the numerator and denominator in such a way as to avoid exponentially small terms. It turns out that this is easiest to do if we compute the *inverse* of the conditional probability, rather than the conditional probability itself. To illustrate, let us write x for the "unobservable" components and y for the "observable" components and (x, y) for the complete vector of variables. The Gibbs distribution has the form

$$P(x, y) = \prod_{c \in \mathcal{C}} F_c((x, y)_c)$$

and, therefore,

$$P(x|y) = \frac{\prod_{c \in \mathcal{C}} F_c((x, y)_c)}{\sum_{\tilde{x} \in \mathcal{R}^n} \prod_{c \in \mathcal{C}} F_c((\tilde{x}, y)_c)}$$

where n is the dimension of x.

The inverse, $1/P(x|y)$, is then

$$\frac{1}{P(x|y)} = \sum_{\tilde{x} \in \mathcal{R}^n} \prod_{c \in \mathcal{C}} \left(\frac{F_c((\tilde{x}, y)_c)}{F_c((x, y)_c)} \right).$$

Our only interest is in efficiently summing over $\tilde{x}$, so let us make things transparent by fixing x and y and writing

$$G_c(\tilde{x}_c) = \frac{F_c((\tilde{x}, y)_c)}{F_c((x, y)_c)}$$

in which case the problem becomes the evaluation of

$$\sum_{\tilde{x} \in \mathcal{R}^n} \prod_{c \in \mathcal{C}} G_c(\tilde{x}_c).$$

The clique structure is the same as we started with, and therefore so is the dynamic programming principle and the number of operations. This time, however, there are no numerical problems. Consequently, we will take this approach when, in Section IV-B, we compute some posterior probabilities of maximum-likelihood decodings.

III. Linear Graphs

Linear dependency graphs come up in many applications: speech recognition, convolutional coding, filtering, and control, among others. In general, there is an observation vector y and a "state vector" x, and a joint dependency structure like the one in Fig. 2(a) or 2(b).

In a speech recognition system, x_t might represent a portion of a phoneme uttered as part of a word, or perhaps even a pair of words, so that the state space is potentially quite large, representing the word or pair of words in addition to the phoneme and phoneme fraction. The observable y_t is some representation or encoding of the associated acoustic signal, or more precisely, the signal as it has been recorded by the microphone. In the speech application, this particular dependency graph comes out of the much-used hidden Markov model, under which

$$P(x_1, \ldots, x_n) = P_1(x_1) \prod_{i=2}^{n} P_i(x_i|x_{i-1})$$

and

$$P(y_1, \ldots, y_n|x_1, \ldots, x_n) = \prod_{i=1}^{n} Q_i(y_i|x_i)$$

or

$$P(y_1, \ldots, y_{n-1}|x_1, \ldots, x_n) = \prod_{i=1}^{n-1} Q_i(y_i|x_i, x_{i+1}).$$

The joint distribution is clearly Gibbs with respect to the graph in Fig. 2(a) or 2(b), depending on which model is used for $P(y|x)$.

The object of interest is, of course, the configuration x, and as we have already noticed, its posterior distribution, given y, corresponds to a simple linear graph. So the computational complexity, whether computing a MAP sequence $x_1, \ldots, x_n$, or the posterior probability of such a sequence, is no worse than $n|\mathcal{R}|^2$ (using a left-to-right or right-to-left visitation schedule). Furthermore, in many applications, most transitions $x_{i-1} \to x_i$

are ruled out, *a priori*, meaning that $P_i(x'|x'') = 0$ for most $x', x'' \in \mathcal{R}$. This can help substantially: if the software can be structured to efficiently ignore the zero transitions, then instead of $n|\mathcal{R}|^2$ operations, we can expect something more like $n \cdot k|\mathcal{R}|$, where k is the "arity" or number of possible transitions from a state $x \in \mathcal{R}$.

A. Convolutional Codes and Memoryless Channels

Convolutional codes also give rise to linear dependency graphs, though the neighborhood structure is generally richer than the nearest neighbor system of first-order Markov processes. We shall formulate, here, convolutional decoding as an instance of dynamic programming for general graphs, and recover the well-known Viterbi algorithm [5], [31]. There is nothing new in this exercise (in particular, see [7] and [9]), but our general viewpoint does suggest some extensions that may be of some practical value. One could, for instance, perform exact maximum-likelihood decoding even when the channel noise is not white. Such Markov dependency within the error process might afford a good model of bursting (see [20]). In this case, the dynamic programming principle still holds and the maximum-likelihood decoding is still computable, at a modest increase in computational cost. Furthermore, the exact posterior probability of the maximum-likelihood decoding is also computable, for about as much additional computation as was used for the decoding itself.

Recall that an (n, k, m) convolutional code is defined through a generator matrix G of the form

$$G = \begin{pmatrix} G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & 0 & \cdots \\ 0 & 0 & G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & \cdots \\ & & & \ddots & \ddots & \ddots & \cdots & \ddots & & \end{pmatrix}$$

where each G_i is a $k \times n$ submatrix. For convenience, we will stick to binary codes, so that the elements of G are in $\{0, 1\}$.

To maintain the connection with Section II, we introduce the following (somewhat unusual) notation: $\mathcal{G}_T$ will be the $(T+1)k \times (T+1)n$ upper-left submatrix of G, $x_i \in \{0, 1\}^k$ will represent the ith block of information bits ($0 \le i \le T$), and $v_i \in \{0, 1\}^n$ will represent the corresponding block of code (output) bits. If $x = (x_0, \ldots, x_T)$ and $v = (v_0, \ldots, v_T)$, then $v = x\mathcal{G}_T$.

Since cliques determine computational complexity, it is useful to observe that

$$\begin{aligned} v_0 &= x_0 G_0 \\ v_1 &= (x_0, x_1) \begin{pmatrix} G_1 \\ G_0 \end{pmatrix} \\ &\vdots \\ v_m &= (x_0, \ldots, x_m) \begin{pmatrix} G_m \\ \vdots \\ G_0 \end{pmatrix} \\ v_{i+m} &= (x_i, \ldots, x_{i+m}) \begin{pmatrix} G_m \\ \vdots \\ G_0 \end{pmatrix}, \qquad 1 \le i \le T - m. \end{aligned}$$

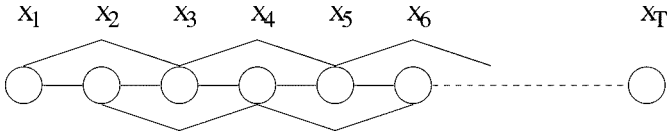

Fig. 4. Second-order posterior dependency of a convolutional code($m = 2$).

Thus, the dependency of v_i on x has the form

$$v_i = F_i(x_{i-m}, x_{i-m+1}, \ldots, x_i)$$

with the understanding that $F_i(Z_0, \ldots, Z_m)$ depends only on the last $i+1$ arguments when $0 \le i \le m$.

Suppose the codeword v goes through a channel C and occasionally gets corrupted. Let v_{ij} be the jth bit of the ith code block ($j \in \{1, \ldots, n\}$, $i \in \{0, \ldots, T\}$) and let y_{ij} be the corresponding output bit. The usual channel model has the form

$$y_{ij} = C(v_{ij}, \eta_{ij})$$

where η_{ij} are independent and identically distributed (i.i.d.), so that

$$P(y_0, \ldots, y_T|v_0, \ldots, v_T) = \prod_{i=0}^{T} \prod_{j=1}^{n} P(y_{ij}|v_{ij}).$$

Hence,

$$\begin{aligned} P(y|x) &= P(y_0, \ldots, y_T|x_0, \ldots, x_T) \\ &= P(y_0, \ldots, y_T|v_0, \ldots, v_T) \\ &= \prod_{i=0}^{T} \prod_{j=1}^{n} P(y_{ij}|v_{ij}) \\ &= \prod_{i=0}^{T} \prod_{j=1}^{n} P(y_{ij}|(F_i(x_{i-m}, \ldots, x_i))_j) \\ &= \prod_{i=0}^{T} H_i(y_i, x_{i-m}, \ldots, x_i) \end{aligned}$$

where

$$H_i(y_i, x_{i-m}, \ldots, x_i) = \prod_{j=1}^{n} P(y_{ij}|(F_i(x_{i-m}, \ldots, x_i))_j).$$

The clique structure, and hence the computational complexity of dynamic programming, would be unchanged by any of a rich collection of prior models on x, governing the arrival of information bits. Since under the uniform prior, the posterior distribution $P(x|y)$ is proportional to $P(y|x)$, maximum-likelihood decoding is the same as MAP decoding, and the (maximal) cliques are of the form $(i-m, i-m+1, \ldots, i)$. For example, the dependency graph for the case $m = 2$ is depicted in Fig. 4. The obvious site visitation schedule is $0, 1, 2, \ldots, T$, which incurs a maximum boundary of m sites and hence a maximum computational cost of $2^{k(m+1)}$ ($|\mathcal{R}| = 2^k$ since $x_i \in \{0, 1\}^k$) at any one site. With T sites, the number of operations needed to compute the MAP (or maximum-likelihood) decoding is, therefore, $O(T \cdot 2^{k(m+1)})$. This is, of course, the well-known Viterbi algorithm.

Suppose $\hat{x}$ ($\hat{v} = \hat{x}\mathcal{G}_T$) turns out to be the maximum-likelihood block of information bits. It would be a good idea to compute the associated probability that $\hat{x}$ ($\hat{v}$) is correct: $P(\hat{x}|y)$. As we have already seen more generally in Section II-C-2, these probabilities are computed by a straightforward adaptation of

the same dynamic programming (Viterbi) algorithm that gave us $\hat{x}$ in the first place.

B. Burst Errors

One way to model burst errors is with a Markov noise process $\eta_1, \eta_2, \ldots, \eta_{nT+n}$ where $\eta_t \in \{0, 1\}$ and $\eta_t = 1$ represents a transmission error ($0 \rightarrow 1$ or $1 \rightarrow 0$). The typical state is presumably "0," but an occasional transition occurs to "1," and there is some tendency to stay at "1." If

$$p_{\alpha\beta} = \text{Prob}(\eta_{t+1} = \beta | \eta_t = \alpha)$$

then the situation can be modeled by making p_{01} (the probability of initiating a burst) very small, and $p_{10} = 1/\mu$, μ being the average burst length. The channel model is then completed by introducing an initializing probability $p^o_\alpha = \text{Prob}(\eta_1 = \alpha)$, in which case the model for $\{\eta_t\}$ is

$$P(\eta_1, \eta_2, \ldots, \eta_{nT+n}) = p^o_{\eta_1} \prod_{t=1}^{nT+n-1} p_{\eta_t, \eta_{t+1}}.$$

Many elaborations are possible (as developed, for example, in [20], and connected to graphical models in [7]), including, for instance, state-dependent bursting in which statistics of $\{\eta_t\}$ depend on the transmitted data $\{v_t\}$, but the Markov model embodied in $p_{\alpha\beta}$ and p^o_α is sensible and, in any case, suitable for illustration.

What are the implications for computing maximum-likelihood (or MAP) decodings? Introduce the error indicators

$$\xi_{ij} = \begin{cases} 0, & \text{if } y_{ij} = v_{ij} \\ 1, & \text{else.} \end{cases}$$

Then

$$\begin{aligned} P(y|x) &= P(y_0, \ldots, y_T | x_0, \ldots, x_T) \\ &= P(y_0, \ldots, y_T | v_0, \ldots, v_T) \\ &= \underbrace{p^o_{\xi_{0,1}} \prod_{j=1}^{n-1} p_{\xi_{0,j}\xi_{0,j+1}}}_{\substack{\text{function of} \\ (y_0, v_0)}} \prod_{i=1}^{T} \underbrace{p_{\xi_{i-1,n}\xi_{i,1}}}_{\substack{\text{function of} \\ (y_{i-1}, y_i, v_{i-1}, v_i)}} \\ &\quad \times \underbrace{\prod_{j=1}^{n-1} p_{\xi_{i,j}\xi_{i,j+1}}}_{\substack{\text{function of} \\ (y_i, v_i)}} \\ &= \prod_{i=1}^{T} G_i(y_{i-1}, y_i, v_{i-1}, v_i) \\ &= \prod_{i=1}^{T} G_i(y_{i-1}, y_i, x_{i-m-1}, \ldots, x_i) \end{aligned}$$

in light of the relation between v_i and x_i.

Evidently, then, the situation is not much different from the simple i.i.d. channel model. This time, the maximal cliques have the form $(i - m - 1, i - m, \ldots, i)$, which is an expansion over the i.i.d. model by only one site, and therefore the most likely decoding and its posterior probability can be computed with $O(T \cdot 2^{k(m+2)})$ operations, or about 2^k times the decoding cost under a white-noise model.

IV. Production Systems and Tree-Structured Graphs

From the computational viewpoint, the primary virtue of linear graphs is that the computational cost of dynamic programming grows linearly with the number of variables, even while the configuration space grows exponentially. More general lattice graphs, such as Z_d with $d \geq 2$ and nearest neighbor interactions, behave differently. A $T \times T \times \cdots \times T$ sublattice of Z_d will achieve a maximum boundary of at least $N_{\max} = T^{d-1}$, no matter what the site visitation schedule. Computation, therefore, grows exponentially (in T) whenever $d \geq 2$.

In between the linear graph and the lattice graph (with $d \geq 2$) are tree-structured graphs, which fortunately also admit site visitation schedules with bounded maximum boundaries. As an example, consider the tree-structured (but cyclic) dependency graph on X in Fig. 5.

Label the sites at level l, from left to right, by s^l_i $1 \leq i \leq 2^{p-l}$ and consider the "bottom-up, left-to-right" site visitation schedule: $s^0_1, s^0_2, \ldots, s^0_{2^p}, s^1_1, s^1_2, \ldots, s^1_{2^{p-1}}, \ldots, s^p_1$. The largest boundary encountered for any connected component has size $N_{\max} = 2$, *and this is independent of p, the depth of the tree*. Since there are $2^{p+1} - 1$ nodes, the number of dynamic programming operations for computing probabilities and most likely configurations of associated Gibbs distributions is $O(2^{p+1}|\mathcal{R}|^3)$. Thus computation grows linearly with the number of variables for Gibbs distributions on tree-structured graphs.

Because of their computational advantages, tree-structured dependencies are attractive modeling tools (e.g., [32], [33]). They also come up naturally when working with *production systems*, which define the so-called "context-free" grammars studied in formal linguistics ([4]).

In this section, we will introduce a suite of error-correcting codes that are based on, or in any case admit representations in terms of, production systems. (As we shall see, the approach turns out to be nothing more than a reformulation of Forney's "Squaring Construction," [21]. See also Gore [34], for an earlier but less developed squaring-type construction.) In computational linguistics, the range of the (vector of) leaf-node variables is known as the "yield" or "language." In our application, the range is a set of permissible codewords rather than a set of well-formed sentences. In either application, whether to linguistics or coding, the tree structure is exploited to design efficient computational algorithms.

In way of illustration, let us examine some of the computational consequences of a formal grammar representation of the even-parity code. A context-free grammar (in "Chomsky normal form"—see [4]) consists of a finite set of nonterminal symbols, $\mathcal{N}$, a start symbol $S \in \mathcal{N}$, a finite set of terminal symbols $\mathcal{T}$, and, for every $A \in \mathcal{N}$, a finite set of production rules, each of the form

$$A \rightarrow BC, \qquad B, C \in \mathcal{N}$$

or

$$A \rightarrow t, \qquad t \in \mathcal{T}.$$

Among the advantages of this scheme is that single and multiple productions can be jointly expressed as

$$B_i^{\boldsymbol{l},n} = \bigcup_{j=0}^{I^l-1} B_{g(x(i)\wedge j,\, z(i))}^{\boldsymbol{l-1},n} \times B_{x(i)\wedge j}^{\boldsymbol{l-1},n}$$

where the auxiliary integers $x(i)$ and $z(i)$ are defined by the correspondences

$$x(i) \leftrightarrow \boldsymbol{x}(i) = \boldsymbol{x}_0\boldsymbol{x}_1 \ldots \boldsymbol{x}_{K(l)}$$

and

$$z(i) \leftrightarrow \boldsymbol{z}(i) = \boldsymbol{z}_0\boldsymbol{z}_1 \ldots \boldsymbol{z}_{K(l)}$$

and the binary operator $\wedge$ is the same as g itself—bitwise exclusive-OR, introduced for notational convenience. In these expressions, $\boldsymbol{z}_k$ is the $\binom{l-1}{k}$-bit binary expansion of $z_k = i_k \bmod m_k^{l-1}$ and $\boldsymbol{x}_k$ is the $\binom{l-1}{k}$-bit expansion of $x_k = \lfloor i_{k+1}/m_{k+1}^{l-1} \rfloor$ (or 0 if $k = K(l)$). Note that for single productions (i.e., $K(l) = l$), the strings $\boldsymbol{x}_{K(l)}$ and $\boldsymbol{z}_{K(l)}$ have length $\binom{l-1}{l} \triangleq 0$ and can, therefore, be ignored; however, for multiple productions ($K(l) < l$) they cannot be ignored.

A further distinction of this scheme is that the multitude of productions for the RM code $\mathrm{RM}\,(p-\alpha, p)$ can be readily computed from a comparatively small set of stored integers—the $2(p+1)$ parameters $\{N^l, I^l | 0 \le l \le p\}$ and the set of $2\sum_{l=1}^{p} N^l$ auxiliary x's and z's, one pair of integers for each state at each level $l \ge 1$. But of far greater consequence is the foundation we have established for constructing a simple system of state-space partitions for thinned RM grammars.

The basic partition at level l is constructed as a succession of binary refinements of the set of N^l states. There are $\log_2 N^l + 1$ coarsenings, $q \in \{0, 1, \ldots, \log_2 N^l\}$, each with $N^{l,q} \triangleq \lfloor N^l/2^q \rfloor$ *super states* denoted by the pair (q, i) at coarseness q. Specifically, we define the coarsened symbols

$$B_i^{\boldsymbol{l},n,q} \triangleq \{B_k^{\boldsymbol{l},n} | \lfloor k/2^q \rfloor = i\}$$

for $0 \le i \le N^{l,q} - 1$.

The extraordinary feature of our choice of super states is that they inherit the underlying structure of the RM grammar. In fact, these coarsened symbols obey the recursive set relation (proved in [36])

$$B_i^{\boldsymbol{l},n,q} = \bigcup_{j=0}^{I^{l,q}-1} B_{g(\overline{x}\wedge j,\, \overline{z})}^{\boldsymbol{l-1},n,\overline{q}} \times B_{\overline{x}\wedge j}^{\boldsymbol{l-1},n,\overline{q}} \qquad (12)$$

where $\overline{x}$ and $\overline{z}$ are simple functions of the auxiliary integers $x(i)$ and $z(i)$, respectively, while $\overline{q}$ and $I^{l,q}$ depend only on the level l and coarseness q. Although this coarsening scheme is somewhat cumbersome to express mathematically, its computational implementation is straightforward and facilitates the remarkably fast CTFDP decoding algorithm presented in Section V-C.

For example, since the hierarchy of coarsened symbols retains the underlying RM structure, the run-time computation of super-state productions required by the CTFDP procedure is trivial. If (q, i), (q_L, L), and (q_R, R) are super states at the respective sites s_i^l, s_{2i-1}^{l-1}, and s_{2i}^{l-1}, then $(q, i) \to [(q_L, L), (q_R, R)]$ is an allowed super-state production if and only if L shares a (suitably sized—see [36]) binary prefix with $g(\overline{x} \wedge j, \overline{z})$ and R shares a binary prefix with $\overline{x} \wedge j$ for some $0 \le j \le I^{(l,q)} - 1$; for if this condition is met, there is an allowed state production contained within the postulated super-state production. We are thus able to compute super-state productions by inspection!

REFERENCES

[1] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[2] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 179–190, 1983.

[3] C. Cannings, E. A. Thompson, and H. H. Skolnick, "Probability functions on complex pedigrees," *Adv. Appl. Probab.*, vol. 10, pp. 26–61, 1978.

[4] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.

[5] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

[6] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *European Trans. Telecommun.*, vol. 6, pp. 513–525, 1995.

[7] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Dept. Elec. Eng., Linköping Univ., Linköping, Sweden, 1996.

[8] G. D. Forney Jr., "On iterative decoding and the two-way algorithm," in *Proc. Intl. Symp. Turbo Codes and Related Topics*, Brest, France, 1997.

[9] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," Department of Electrical and Computer Engineering, University of Toronto, Tech. Rep., 1998.

[10] R. Dechter, "Bucket elimination: A unifying framework for probabilistic inference," in *Learning in Graphical Models*, M. I. Jordan, Ed. Cambridge, MA: MIT Press, 1999, pp. 75–104.

[11] J. Pearl, *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Kaufmann, 1988.

[12] S. L. Lauritzen, *Graphical Models*. Oxford, U.K.: Oxford Univ. Press, 1996.

[13] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1996.

[14] R. G. Gallager, "Low-density parity-check codes," Ph.D. dissertation.

[15] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, Sept. 1981.

[16] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[17] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. Int. Conf. Communications (ICC'93)*, Geneva, Switzerland, 1993, pp. 1064–1070.

[18] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[19] F. Kschischang and B. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 219–230, Feb. 1998.

[20] L. N. Kanal and A. R. K. Sastry, "Models for channels with memory and their applications to error control," in *Proc. IEEE*, vol. 66, 1978, pp. 724–744.

[21] G. D. Forney Jr., "Coset codes—Part II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1152–1187, Sept. 1988.

[22] C. S. Raphael, "Coarse-to-fine dynamic programming," *IEEE Trans. Pattern Anal. Machine Intell.*, to be published.

[23] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, pp. 257–286, 1989.

[24] Y. Amit and A. Kong, "Graphical templates for model registration," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, pp. 225–236, 1996.

[25] B. J. Frey, *Graphical Models for Machine Learning and Digital Communication*. Cambridge, MA: MIT Press, 1998.

[26] J. Hammersley and P. Clifford, "Markov fields on finite graphs and lattices," Univ. California, Berkeley, Tech. Rep., 1968.

[27] G. Winkler, *Image Analysis, Random Fields, and Dynamic Monte Carlo Methods: A Mathematical Introduction*. New York: Springer–Verlag, 1995.

[28] G. Kallianpur, *Stochastic Filtering Theory*. New York: Springer-Verlag, 1980.
[29] H. Künsch, S. Geman, and A. Kehagias, "Hidden Markov random fields," *Ann. Appl. Probab.*, vol. 5, pp. 577–602, 1995.
[30] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM J. Alg. Discr. Meth.*, vol. 8, pp. 277–284, 1987.
[31] G. D. Forney Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, 1973.
[32] P. W. Fieguth and A. S. Willsky, "Fractal estimation using models on multiscale trees," *IEEE Trans. Signal Processing*, vol. 44, pp. 1297–1300, May 1996.
[33] M. Meila, "Learning with mixtures of trees," Ph.D. dissertation, MIT, Cambridge, MA, 1999.
[34] W. C. Gore, "Further results on product codes," *IEEE Trans. Inform.Theory*, vol. IT-16, pp. 446–451, July 1970.
[35] R. Kindermann and J. Snell, *Markov Random Fields and Their Applications*. Providence, RI: Amer. Math. Soc., 1980.
[36] K. Kochanek, "Dynamic programming algorithms for maximum likelihood decoding," Ph.D. dissertation, Div. Appl. Math., Brown Univ., Providence, RI, 1998.
[37] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984.
[38] A. Kam and G. Kopec, "Document image decoding by heuristic search," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, pp. 945–950, Sept. 1996.
[39] ——, "The iterated complete path algorithm," Xerox Palo Alto Research Center, Tech. Rep., 1995.