3 Factorizing Shortest Paths with Randomized Optimum Models

Daniel Tarlow dtarlow@microsoft.com
Alexander Gaunt t-algaun@microsoft.com
Microsoft Research
Cambridge, UK

Ryan Adams rpa@seas.harvard.edu
Harvard University and Twitter
Cambridge, MA, USA

Richard S. Zemel zemel@cs.toronto.edu
University of Toronto
Toronto, ON, Canada

Randomized Optimum models (RandOMs) are probabilistic models that define distributions over structured outputs by making use of structured optimization procedures within the model definition. This chapter reviews RandOMs and develops a new application of RandOMs to the problem of factorizing shortest paths; that is, given observations of paths that users take to get from one node to another on a graph, learn edge-specific and user-specific trait vectors such that inner products of the two define user-specific edge costs, and the distribution of observed paths can be explained as users taking shortest paths according to noisy samples from their cost function.

3.1 Introduction

A broad challenge in statistics and machine learning is to build probabilistic models of structured data. This includes abstract structures like segmentations, colorings, matchings, and paths on graphs, and natural structures like images, text, source code, and chemical molecules. The main difficulty is that estimating the normalizing constant for commonly-used modeling distributions over these objects is often computationally hard. An interesting computational phenomenon is that in some cases where it is challenging to compute a sum over the entire space, it is efficient to find the maximum (or minimum). For example, the problem of computing a matrix permanent, which is #-P hard (Valiant, 1979), corresponds to computing a normalizing constant for a probabilistic model where the most probable configuration can be computed efficiently as a bipartite matching. More specifically, given an energy function $f(\cdot)$ over structures $\boldsymbol{y}$ (e.g., a path on a graph $\mathcal{G}$) from an output space $\mathcal{Y}$ (e.g., all paths on $\mathcal{G}$), the normalizing constant or *partition function* is $Z = \sum_{\boldsymbol{y}\in\mathcal{Y}} \exp\{-f(\boldsymbol{y})\}$. This chapter focuses on the case where the output space is a combinatorial set, by which we mean that membership can be tested efficiently but enumeration is intractable (Bouchard-Côté and Jordan, 2010); however, in principle, the output space could also be continuous. The corresponding optimization problem is to find the most probable structure: $\operatorname{argmin}_{\boldsymbol{y}\in\mathcal{Y}} f(\boldsymbol{y})$.

The typical approach for defining probability distributions over structured objects is to use a *Gibbs distribution*. That is, make sensible assumptions about the structure of an energy function $f(\boldsymbol{y})$ and combinatorial set $\mathcal{Y}$, and then define $p(\boldsymbol{y}) \propto 1\{\boldsymbol{y} \in \mathcal{Y}\} \exp\{f(\boldsymbol{y})\}$. For example, to define a model of foreground-background segmentations of an image with D pixels, a common choice might be $\boldsymbol{y} \in \{0,1\}^D$ and to define an energy function according to a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as

$$f(\boldsymbol{y}) = f(\boldsymbol{y}; g) = \sum_{i\in\mathcal{V}} g_i \cdot y_i + \sum_{ij\in\mathcal{E}} g_{ij} \cdot 1\{y_i = y_j\},$$

which encodes the assumption that there are node-specific costs g_i for each pixel i to be labeled 1 (foreground) and that edges in the graph encourage neighboring nodes to take on the same label with an edge-dependent cost for differing g_{ij}. A typical choice of edge structure would be a 4-connected grid, where there are edges between nearest neighbor pixels.

While the above assumptions are sensible, they immediately lead to computational difficulty. Consider making test-time predictions, which depend on $p(\boldsymbol{y})$ and therefore require the intractable Z. There are two common choices: (1) use approximate inference like belief propagation (see e.g., Koller and Friedman (2009)) to compute approximate marginals, or (2) use Markov chain Monte Carlo (MCMC) to draw approximate samples (see e.g., (Robert and Casella, 2013)). The focus of this chapter is on cases where the combinatorial structure of the object is important, so marginals do not suffice. The point of Randomized Optimum Models (RandOMs) is to provide an efficient alternative to MCMC at test time without sacrificing the well-founded probabilistic model.

The outline is as follows:

- Background on structured prediction, and design considerations for building probabilistic models of structured objects
- A review of RandOMs
- Shortest Path Factorization with RandOMs
- Experiments
- Related work
- Discussion

3.2 Building Structured Models: Design Considerations

Structured prediction is a large field, and there are many approaches for learning models of structured objects. This section describes a high level overview of the key considerations, with a bias towards probabilistic models of structured objects.

A key issue that affects the choice of model is what the utility function will be. That is, how will we evaluate the quality of a test-time output? Is the system going to be used by some downstream process, or is it going to be used to make a single prediction? In the former case, a natural output for the system is a probability distribution (e.g., a probability that a patient has cancer); in the latter case, the utility function needs to be considered by the system (e.g., how unpleasant the patient finds the treatment, and how much value they would place on being cured).

A second question is about the structure of the utility function, which is relevant even if the system is producing a probability distribution, because it has bearing on how the probability distribution should be represented. In a structured prediction setting, a key property of utility functions to consider is whether they are sensitive to high order structure or not. For example, if an image segmentation system is judged based on the number of pixel-level classifications that it gets correct, then the utility function depends only on low order statistics of the output probability distribution, i.e., it can be shown that the expected utility of a predictive distribution depends only on the marginal distributions of each pixel's label. In this case, representing a probability distribution over pixel labelings as a set of marginal distributions is perfectly reasonable. Even in cases where the utility function appears at first glance to have high order interactions, such as with the intersection-over-union measure that is common in evaluating image segmentations (Everingham et al., 2010), Nowozin (2014) has shown that marginal distributions contain enough information to make accurate utility-aware predictions.

However, there are cases where the utility function truly is high order, and in fact, these are very common cases. One might even argue that *most* natural utility functions over structured objects depend heavily on high order structure, and it is only computational convenience that leads to utility functions based on low order structure. Examples of utility functions that depend on high order structure include perceptual measures of the naturalness of an image or image segmentation when outputting images or pixel-wise labels (Movahedi and Elder, 2010; Lubin, 1998; Wang et al., 2004), measures of whether code compiles when outputting source code (Nguyen et al., 2014), measures of the meaningfulness of a generated sentence when outputting language, and measures of whether a driver could follow a path that is output by the model.

When the utility function has high order structure and we wish to directly output a single prediction, then in some cases max-margin learning (Taskar et al., 2004; Tsochantaridis et al., 2005) can be a good option. High order utility functions present challenges, but can sometimes be handled efficiently, such as in certain image segmentation settings (Tarlow and Zemel, 2012; Pletscher and Kohli, 2012).

When the utility function has high order structure and we wish to output a probability distribution, sample-based representations of the output distribution are the natural choice. This is the setting that motivates RandOMs, along with several other works, including some in this book, such as Perturb & MAP (Chapter 2), PAC-Bayesian perturbation models (Chapter 10), and MAP-perturbation models (Chapter 5); see Section 3.9 for a discussion of the similarities and differences between RandOMs and other works that focus on this regime. Our focus is to train models such that at test time, we can efficiently produce perfect samples from the model without resorting to MCMC or rejection sampling.

3.3 Randomized Optimum Models (RandOMs)

This section introduces notation and then develops the RandOM model.

RandOMs implicitly define a probability distribution over an output space $\mathcal{Y}$ via a generative procedure that includes a call to an algorithm that performs optimization over $\mathcal{Y}$. In the typical instantiation, $\mathcal{Y}$ is a combinatorial set and the optimization algorithm is a discrete optimization procedure.

3.3.1 Notation.

Let $f_{\boldsymbol{w}} : \mathcal{Y} \to \mathbb{R}$ be a family of scoring functions indexed by $\boldsymbol{w} \in \mathbb{R}^P$, each of which maps a structure $\boldsymbol{y}$ to a real-valued cost. Let $\mathcal{Y}$ be the set of legal structures. For example, $\boldsymbol{w}$ may be node weights for a weighted vertex cover algorithm or edge costs for a graph cut algorithm, and $\mathcal{Y}$ would be the set of all vertex covers or the set of all graph cuts, respectively. In these cases, the individual dimensions of $\boldsymbol{w}$ might be costs of specific nodes or edges in some graph. A further description of f's dependence on $\boldsymbol{w}$ appears below.

It will then be useful to define $F : \mathbb{R}^P \to \mathcal{Y}$ as the function that executes an optimization algorithm given parameters $\boldsymbol{w}$ and returns a cost-minimizing configuration $\boldsymbol{y}^*$; i.e., $F(\boldsymbol{w}; \mathcal{Y}) = \operatorname{argmin}_{\boldsymbol{y} \in \mathcal{Y}} f_{\boldsymbol{w}}(\boldsymbol{y})$. Also useful will be the *inverse set* $F^{-1}(\boldsymbol{y}; \mathcal{Y})$, which is defined as $F^{-1}(\boldsymbol{y}; \mathcal{Y}) = \{\boldsymbol{w} \mid F(\boldsymbol{w}; \mathcal{Y}) = \boldsymbol{y}\}$. When the appropriate $\mathcal{Y}$ is clear from context, it will be dropped from the notation, resulting in $F(\boldsymbol{y})$ and $F^{-1}(\boldsymbol{y})$.

In some problems there is a notion of *legal* settings of $\boldsymbol{w}$. For example, a shortest path algorithm might reasonably assert that all edge costs should be non-negative, or a graph cut algorithm may assert that edge potentials are submodular. To handle these cases, the predicate $\mathcal{L} : \mathbb{R}^P \to \{0, 1\}$ will be used to indicate whether a $\boldsymbol{w}$ is legal.

3.3.2 RandOM Model.

The key idea of RandOM models is to define probabilistic models where parameters $\boldsymbol{w}$ are latent variables. That is, a probabilistic model $p(\boldsymbol{y}; \psi)$ is defined via a distribution over $\boldsymbol{w}$, parameterized by ψ; the link between $\boldsymbol{y}$ and $\boldsymbol{w}$ values is a deterministic relationship that comes from running the optimization algorithm:

$$p(\boldsymbol{y}; \psi) \propto \int p(\boldsymbol{w}; \psi)\, 1\{F(\boldsymbol{w}) = \boldsymbol{y}\}\, 1\{\mathcal{L}(\boldsymbol{w})\}\, d\boldsymbol{w}. \tag{3.1}$$

The design space of distributions over $\boldsymbol{w}$ is large and flexible. Many variations are possible, such as conditioning on inputs $\mathbf{x}$:

$$p(\boldsymbol{y} \mid \mathbf{x}; \psi) \propto \int p(\boldsymbol{w} \mid \mathbf{x}; \psi)\, 1\{F(\boldsymbol{w}) = \boldsymbol{y}\}\, 1\{\mathcal{L}(\boldsymbol{w})\} d\boldsymbol{w}, \tag{3.2}$$

which is the form that was the focus of Tarlow et al. (2012). It would also be straightforward to treat ψ as random variables which themselves have prior distributions. The key to test-time tractability is that a sample from $p'(\boldsymbol{w}) \propto p(\boldsymbol{w})1\{\mathcal{L}(\boldsymbol{w})\}$ can be drawn efficiently. Given the sample of $\boldsymbol{w}$, the optimization algorithm can be executed to yield a sample $\boldsymbol{y}$; i.e., set $\boldsymbol{y} = F(\boldsymbol{w})$.

3.3.3 Constructing Conditional Random Field-like f

This section describes a pattern for constructing $p(\boldsymbol{w} \mid \mathbf{x}; \psi)$ that parallels the energy function used in conditional random field (CRF) models. To illustrate how this works within the RandOM formulation, we focus on a pairwise CRF with binary variables, as would be used for the foreground-background segmentation example in the introduction.

To review, CRFs define distributions over $\mathcal{Y}$ via the Gibbs distribution. For pairwise CRFs with binary variables, the energy function $f(\boldsymbol{y})$ is constructed as a sum of unary and pairwise terms:

$$f(\boldsymbol{y}) = \sum_{i \in \mathcal{V}} g_i(y_i, \mathbf{x}; \psi) + \sum_{ij \in \mathcal{E}} g_{ij}(y_i, y_j, \mathbf{x}; \psi). \tag{3.3}$$

The $g(\cdot)$ terms are parameterized by weights ψ and can depend arbitrarily on the input $\mathbf{x}$, but have only local dependence on $\boldsymbol{y}$. The $g(\cdot)$ functions are usually constructed as a weighted sum of unary features and pairwise features. An example unary feature would be an affinity for the average color of image $\mathbf{x}$ around pixel i to class y_i. An example pairwise feature would be a cost for neighboring pixels i and j to take different classes with strength depending on the difference of appearance of the pixels.

Finally, the probability of a configuration is defined by the Gibbs distribution: $p(\boldsymbol{y}) \propto \exp\{-f(\boldsymbol{y})\}$.

3.3.3.1 CRF Energy Functions in RandOM Notation

First, a vector of *sufficient statistics* of $\boldsymbol{y}$ are chosen, denoted $\boldsymbol{\rho}(\boldsymbol{y}) = (\rho_p(\boldsymbol{y}))_{p=1}^{P}$ where $\rho_p : \mathcal{Y} \to \{0, 1\}$. Each $\rho_p(\cdot)$ is an indicator function that selects out some statistic of $\boldsymbol{y}$ that is relevant for the model. Example indicator functions are whether a particular subset of dimensions of $\boldsymbol{y}$ take on a particular joint configuration, or they could indicate whether the number of dimensions of y taking on a particular value (say a) is equal to some value (say b); i.e., $\rho_p(\boldsymbol{y}) = 1\{(\sum_i 1\{y_i = a\}) = b\}$.

As another example, in a pairwise graphical models, there are unary and pairwise sufficient statistics. The unary sufficient statistics are functions indicating if $y_i = a$ for each variable i and each possible value a. Pairwise sufficient statistics are defined over all edges and might indicate all joint configurations of a pair of neighboring variables, i.e.,

$(1\{y_i = 0 \wedge y_j = 0\}, \cdots, 1\{y_i = 1 \wedge y_j = 1\})$, or just whether neighboring variables take on the same label, i.e., $(1\{y_i = y_j\})$. There is flexibility in the choice of sufficient statistics. The main issue to be mindful of is that the choice of sufficient statistics can impact the tractability of the minimization problem, so some care must be taken. More examples of choices of sufficient statistics that lead to tractable optimization appear below.

Given a vector of sufficient statistics, the definition of $f_{\boldsymbol{w}}(\boldsymbol{y})$ is then simply that each dimension of $\boldsymbol{w}$ weights the sufficient statistic in the corresponding dimension:

$$f_{\boldsymbol{w}}(\boldsymbol{y}) = \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y}). \tag{3.4}$$

To produce an equivalent $f_{\boldsymbol{w}}$ using the RandOM formulation, define $p(\boldsymbol{w} \mid \mathbf{x}; \boldsymbol{\psi})$ to be a deterministic function of input $\mathbf{x}$ and parameters $\boldsymbol{\psi}$ as follows.

First, rewrite f as

$$f(\boldsymbol{y}) = \sum_{i \in \mathcal{V}} \sum_{\hat{y}_i} 1\{y_i = \hat{y}_i\} g_i(\hat{y}_i, \mathbf{x}; \psi) \tag{3.5}$$

$$+ \sum_{ij \in \mathcal{E}} \sum_{\hat{y}_i, \hat{y}_j} 1\{y_i = \hat{y}_i \wedge y_j = \hat{y}_j\} g_{ij}(\hat{y}_i, \hat{y}_j, \mathbf{x}; \psi). \tag{3.6}$$

Then it becomes clear that by defining sufficient statistics vector $\boldsymbol{\rho}(\boldsymbol{y})$ to be a concatenation of $(1\{y_i = a\})$ for all i and a with $(1\{y_i = 0 \wedge y_j = 0\}, 1\{y_i = 0 \wedge y_j = 1\}, 1\{y_i = 1 \wedge y_j = 0\}, 1\{y_i = 1 \wedge y_j = 1\})$ for all $ij \in \mathcal{E}$, and analogously defining $\boldsymbol{g}$ to be a vector of the $g_i(\cdot)$ or $g_{ij}(\cdot)$ functions corresponding to the entries of $\boldsymbol{\rho}(\boldsymbol{y})$, then setting $\boldsymbol{w} = \boldsymbol{g}$ ensures that $f(\boldsymbol{y}) = f_{\boldsymbol{w}}(\boldsymbol{y})$ for all $\boldsymbol{y}$.

Of course, if $\boldsymbol{w}$ is a deterministic function of $\mathbf{x}$ and $\boldsymbol{\psi}$, then the output distribution will be degenerate and assign nonzero probability to a single $\boldsymbol{y}$. Instead, to induce a meaningful distribution over outputs, $\boldsymbol{w}$ must be random. This is in contrast to CRFs, which define an energy function to be deterministically constructed from inputs, but then the distribution over $\boldsymbol{y}$ given the energy function is random.

3.3.3.2 Example: The Gibbs Distribution

As noted by Papandreou and Yuille (2011) and extended by Hazan and Jaakkola (2012), it is possible to leverage properties of Gumbel distributions in order to exactly represent the Gibbs distributions that arises in standard CRF models. While this connection is of theoretical interest, it is not a practical construction because it requires the set of sufficient statistics to be exponentially large, with one sufficient statistic for each possible configuration of $\boldsymbol{y}$. The connection is presented here for completeness. See Chapters 2, 6 and 7 for additional discussions of related issues.

A random variable G is said to have a Gumbel distribution with location $m \in \mathbb{R}$ if the CDF is $p(G < g) = \exp\left(-\exp(-g+m)\right)$. The key property of Gumbel distributions is that for a collection of independent Gumbels $G_1, \ldots, G_K$ with locations $m_1, \ldots, m_K$ respectively, the distribution of the maximum is also Gumbel-distributed but with location equal to the logsumexp of the locations, and the argmax is distributed according to the Gibbs distribution where m_k is the negative energy of configuration k. More precisely,

$$\max_k G_k \sim \text{Gumbel}\left(\log \sum_{k=1}^{K} \exp m_k\right), \text{ and} \tag{3.7}$$

$$\operatorname*{argmax}_k G_k \sim \frac{\exp m_k}{\sum_{k'=1}^{K} \exp m_{k'}}. \tag{3.8}$$

Letting $p = 1, \ldots, |\mathcal{Y}|$ index all configurations and $\hat{\boldsymbol{y}}(p)$ be the p^{th} configuration under this ordering, we can then let $\boldsymbol{\rho}(\boldsymbol{y}) = (1\{\boldsymbol{y} = \hat{\boldsymbol{y}}(p)\})_{p=1}^{|\mathcal{Y}|}$; i.e., there is one sufficient statistic for each $\hat{\boldsymbol{y}} \in \mathcal{Y}$ indicating whether $\boldsymbol{y}$ is exactly equal to $\hat{\boldsymbol{y}}(p)$. Finally, let $\bar{\boldsymbol{w}} = (-f(\hat{\boldsymbol{y}}(p)))_{p=1}^{|\mathcal{Y}|}$ be the vector that puts the negative energy of configuration p in dimension p, and let $-w_p \sim \text{Gumbel}(\bar{w}_p)$ for all p. Then

$$\operatorname*{argmin}_p \boldsymbol{w}^\top \boldsymbol{\rho}(\hat{\boldsymbol{y}}(p)) = \operatorname*{argmax}_p -w_p \sim \frac{\exp \bar{w}_p}{\sum_{p'} \exp \bar{w}_{p'}} = \frac{\exp -f(\hat{\boldsymbol{y}}(p))}{\sum_{p'} \exp -f(\hat{\boldsymbol{y}}(p'))}, \tag{3.9}$$

which shows the equivalence to the Gibbs distribution.

3.3.3.3 Example: Bipartite Matching f

The weighted perfect bipartite matching problem is defined in terms of a bipartite graph $\mathcal{G}$ with partite sets A and B with $J = |A| = |B|$. The only edges in $\mathcal{G}$ are between a node $v \in A$ and $v' \in B$; we will additionally assume that all possible edges exists, so there is an edge from each $v \in A$ to each $v' \in B$.

A perfect matching is a one-to-one mapping between nodes in A and nodes in B. Each edge (v, v') is assigned a cost $w_{vv'}$, and the cost of a matching is the sum of the costs of edges that are included in the matching.

To formalize this in terms of above notation, let $y_{vv'} \in \{0, 1\}$ be an indicator that edge (v, v') is used in a matching. Let $\boldsymbol{y}$ be an ordered list of indicators for each edge $\{y_{vv'} : v \in A, v' \in B\}$. Let $\boldsymbol{w}$ be an analogous ordered list $\{w_{vv'} : v \in A, v' \in B\}$ such that element p of $\boldsymbol{w}$ is the weight for edge being indicated by element p of $\boldsymbol{y}$. Finally, let $\mathcal{Y}$ be the set of binary vectors of length J^2 that correspond to valid matchings according to the encoding of $\boldsymbol{y}$ above. Then the cost of any matching $\boldsymbol{y} \in \mathcal{Y}$ is simply $f_{\boldsymbol{w}}(\boldsymbol{y}) = \boldsymbol{w}^\top \boldsymbol{y}$. (To match the general form in (3.4), $\boldsymbol{\rho}$ could be set to be the identity $\boldsymbol{\rho}(\boldsymbol{y}) = \boldsymbol{y}$, and then $f_{\boldsymbol{w}}(\boldsymbol{y}) = \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y})$ as above.)

3.3.3.4 Example: Shortest Paths f

An encoding of a shortest paths problem is similar. The shortest path problem is defined in terms of a weighted graph $\mathcal{G}$, and a start-end node pair (s, t). The combinatorial problem is to find the shortest path in $\mathcal{G}$ from s to t, where the cost of a path is a sum of the costs of the edges traversed by the path.

To encode an f function corresponding to this problem, let $\boldsymbol{y}$ be a vector of indicators of edges (as above), with dimension p indicating whether edge p is used in the path. Let $\boldsymbol{w}$ be the corresponding vector of edge costs. Then as in the bipartite matching case, $f_{\boldsymbol{w}}(\boldsymbol{y}) = \boldsymbol{w}^\top \boldsymbol{y}$.

The combinatorial set $\mathcal{Y} = \mathcal{Y}(s, t)$ is the set of all simple paths from s to t (i.e., paths with no repeating vertices).

3.3.4 Other types of f

In all of the above examples, f has been a linear function of $\boldsymbol{w}$. It is always possible to define $\boldsymbol{\rho}(\boldsymbol{y}) = (1\{\boldsymbol{y} = \hat{\boldsymbol{y}}\})_{\hat{\boldsymbol{y}} \in \mathcal{Y}}$, and thus if $f_{\boldsymbol{w}}(\boldsymbol{y}) = \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y})$ then each $\boldsymbol{y} \in \mathcal{Y}$ has an independent entry of $\boldsymbol{w}$ and all possible energy functions can be expressed; this is the equivalent of representing an energy function in a tabular form that assigns some cost to each configuration.

While such a construction is as flexible as possible, it does not mean that all interesting $f_{\boldsymbol{w}}(\boldsymbol{y})$ are linear functions of $\boldsymbol{w}$. Indeed, for $F(\boldsymbol{w})$ to be implemented efficiently, $\boldsymbol{w}$ must be represented in some compact form (such as edge costs in the above example), and each efficient combinatorial optimization routine expects an input of a particular form.

3.3.4.1 Example: Connected Components f

For example, consider the weighted connected components problem. Given a weighted graph $\mathcal{G}$, cut all edges with weight less than some parameter τ to get an unweighted graph $\mathcal{G}'$ that contains the uncut edges in $\mathcal{G}$, then partition the nodes into connected components; that is, two nodes v and v' are in the same connected component iff there is a path from v to v' in $\mathcal{G}'$.

For a given τ, the natural parameterization of the problem is to have one dimension of $\boldsymbol{w}$ to represent each edge cost in $\mathcal{G}$. There is some flexibility in how to represent $\boldsymbol{y}$, but one reasonable choice is to let $y_i \in \{1, \ldots, |\mathcal{V}|\}$ be equal to the smallest index j such that nodes i and j are in the same connected component. Then $\mathcal{Y}$ is the set of all $\boldsymbol{y}$ such that all nodes with a given label l are connected via edges where both endpoints are labeled l. One might then ask if there is some choice of sufficient statistics $\boldsymbol{\rho}$ such that $f_{\boldsymbol{w}}(\boldsymbol{y}) = \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y})$ and $\operatorname{argmin}_{\boldsymbol{y}} f_{\boldsymbol{w}}(\boldsymbol{y})$ gives the same output as the connected components algorithm described above. It turns out that this is not possible.

Lemma 3.1. *Let $G : \mathbb{R}^{|\mathcal{E}|} \to \mathcal{Y}$ be the function that maps $\boldsymbol{w}$ to the solution to the above weighted connected component problem with parameter τ. There is no choice of sufficient statistics $\boldsymbol{\rho}(\boldsymbol{y})$ such that for all $\boldsymbol{w}$,* $\operatorname{argmin}_{\boldsymbol{y}} \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y}) = G(\boldsymbol{w}; \tau)$.

Proof. (By contradiction). Suppose there were a choice of $\boldsymbol{\rho}(\boldsymbol{y})$ such that for all $\boldsymbol{w}$, $\operatorname{argmin}_{\boldsymbol{y}} \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y}) = G(\boldsymbol{w})$. Then $F^{-1}(\boldsymbol{y})$ is an intersection of halfspaces $\{\boldsymbol{w} : \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y}) \leq \boldsymbol{w}^\top \boldsymbol{\rho}(\boldsymbol{y'})\}$ for each $\boldsymbol{y'} \in \mathcal{Y}$, and is thus a convex set. However, $G^{-1}(\boldsymbol{y})$ is not a convex set, and thus F cannot be equivalent to G.

To see that $G^{-1}(\boldsymbol{y})$ is not a convex set, consider the fully connected graph on three vertices $1, 2, 3$ with edges $(1, 2), (1, 3), (2, 3)$ and $\tau = 1 - \epsilon$. Let $\boldsymbol{w}^A = (1, 1, 0)$, $\boldsymbol{w}^B = (0, 1, 1)$, and $\boldsymbol{y}^*$ be the configuration where all nodes belong to a single connected component. Clearly $\boldsymbol{w}^A \in G^{-1}(\boldsymbol{y}^*)$ and $\boldsymbol{w}^B \in G^{-1}(\boldsymbol{y}^*)$. However, consider $\boldsymbol{w}^C = \frac{1}{2}\boldsymbol{w}^A + \frac{1}{2}\boldsymbol{w}^B = (.5, 1, .5)$. $G(\boldsymbol{w}^C)$ assigns node 2 to its own connected component, and thus $\boldsymbol{w}^C \notin G^{-1}(\boldsymbol{y})$ and $G^{-1}(\cdot)$ is not always a convex set. □

3.4 Learning RandOMs

There are two main approaches to learning RandOMs. Both are based on an Expectation Maximization (EM) algorithm (Dempster et al., 1977) with $\boldsymbol{w}$ as latent variables. A fully Bayesian treatment would also be straightforward, in which case the M step in the Monte Carlo EM variant would be replaced with an MCMC update, but this approach is not discussed further.

The difference between the two EM approaches is how distributions over $\boldsymbol{w}$ are estimated. In the Monte Carlo EM algorithm (MCEM) (Wei and Tanner, 1990), values of $\boldsymbol{w}$ are sampled from a posterior distribution over $\boldsymbol{w}$; in the Hard EM algorithm, a single most likely estimate of $\boldsymbol{w}$ is used.

In more detail, the EM algorithm can be understood as optimizing a single objective (Neal and Hinton, 1998) via an alternating maximization scheme. In the case of general RandOMs (3.1), the objective given a data set $\mathcal{D} = \{\boldsymbol{y}^{(n)}\}_{n=1}^{N}$ is $J(\psi, \{Q^{(n)}\}_{n=1}^{N})$

$$= \sum_{n=1}^{N} \mathbb{E}_{\hat{\boldsymbol{w}} \sim Q^{(n)}(\cdot)} \left[\log \left(p(\boldsymbol{y}^{(n)} \mid \hat{\boldsymbol{w}}) p(\hat{\boldsymbol{w}}; \psi) \right) - \log Q^{(n)}(\hat{\boldsymbol{w}}) \right] \tag{3.10}$$

$$= \sum_{n=1}^{N} \mathbb{E}_{\hat{\boldsymbol{w}} \sim Q^{(n)}(\cdot)} \left[\log 1\{\boldsymbol{y}^{(n)} = F(\hat{\boldsymbol{w}})\} + \log p(\hat{\boldsymbol{w}}; \psi) - \log Q^{(n)}(\hat{\boldsymbol{w}}) \right]. \tag{3.11}$$

EM algorithms alternate between maximizing J with respect to $\{Q^{(n)}\}_{n=1}^{N}$ (E step) and with respect to ψ (M step). Note that the E step is amenable to embarassingly parallel computation.

3.4.1 M Step

In both the MCEM and Hard EM algorithms, $Q^{(n)}(\cdot)$ is represented via a set of L samples $\hat{\boldsymbol{w}}^{(n1)}, \ldots, \hat{\boldsymbol{w}}^{(nL)}$. The M step is an incremental M step (Neal and Hinton, 1998), meaning that rather than updating ψ to optimality, an update is made that just increases J. Note that given fixed samples from $\{Q^{(n)}\}_{n=1}^{N}$ where each sample is in the corresponding inverse set $F^{-1}(\boldsymbol{y}^{(n)})$, the M step objective (dropping terms that do not depend on ψ) is

$$\sum_{n=1}^{N} \frac{1}{L} \sum_{l=1}^{L} \log p(\hat{\boldsymbol{w}}^{(nl)}; \psi). \tag{3.12}$$

This is a standard maximum likelihood objective with parameters ψ and data $\boldsymbol{w}^{(nl)}$, which can be optimized with whatever standard optimizer is most appropriate for the specific form of $p(\boldsymbol{w}; \psi)$ that is chosen. For example, if $p(\boldsymbol{w}; \psi)$ is a neural network, then stochastic gradient ascent can be used.

3.4.2 Monte Carlo E Step

The optimal choice for $Q^{(n)}(\cdot)$ in the E step is to set it equal to the posterior distribution $p(\boldsymbol{w} \mid \boldsymbol{y}^{(n)}; \psi) \propto p(\boldsymbol{w}; \psi) 1\{F(\boldsymbol{w}) = \boldsymbol{y}^{(n)}\} \mathcal{L}(\boldsymbol{w})$. For most RandOMs, it does not appear possible to represent this posterior in closed form. Instead, in Monte Carlo EM, $Q^{(n)}(\cdot)$ is represented via a set of L samples from this posterior. In principle, any MCMC method can be used in the E step, but Slice Sampling (Neal, 2003) is particularly well suited to handle the structure of the problem, as will be discussed in more detail in Section 3.7.3.

3.4.3 Hard E Step

In the Hard EM algorithm, sampling from the posterior is replaced with a maximization step: $\hat{\boldsymbol{w}}$ is chosen so as to be the $\operatorname{argmax}_{\boldsymbol{w}} p(\boldsymbol{w}; \psi) 1\{F(\boldsymbol{w}) = \boldsymbol{y}^{(n)}\} \mathcal{L}(\boldsymbol{w})$. When $f_{\boldsymbol{w}}(\boldsymbol{y})$ is a linear function of $\boldsymbol{w}$ (as in (3.4)) and $p(\boldsymbol{w}; \psi)$ is a Gaussian distribution (log quadratic), then the argmax computation is a quadratic program (QP). More details of this approach appear in Tarlow et al. (2012). An improved Hard EM algorithm appears in Gane et al. (2014).

3.5 RandOMs for Image Registration

In Tarlow et al. (2012), RandOMs are applied to registration problems. The main application is deformable image registration in volumetric CT scans of human lungs. For each human subject in the data set, data consists of scans at different stages of the respiratory process that are annotated with landmarks. The problem is to take a pair of images with their associated landmarks and determine the correspondences between landmarks across the two images. To formulate this problem as a RandOM, $\mathcal{Y}$ is the set of all bipartite matchings with the first (second) partite set being landmarks in the first (second) image. The sufficient statistics indicate whether landmark i in the first image matches to landmark j in the second image, and $\boldsymbol{w}$ assigns a cost for each i, j pair. Features are extracted for each pair based on the difference in appearance of the volume around the landmarks, and parameters ψ weight the importance of different features.

Experimentally, RandOMs are compared against a Structural SVM approach (Taskar et al., 2004; Tsochantaridis et al., 2005) and Perturb & MAP (Papandreou and Yuille, 2011). Results show that RandOMs are competitive with the alternatives and perform best in terms of accuracy.

3.6 Shortest Path Factorization

This section introduces the problem studied in detail in this chapter.

The Shortest Path Factorization (SPF) problem is to observe a data set $\mathcal{D}$ of pairs of driver ids and paths $\mathcal{D} = \{(d_n, \boldsymbol{y}_n)\}_{n=1}^{N}$ where each $\boldsymbol{y}_n$ is a path through a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $d_n \in \{1, \ldots, D\}$ denotes the identity of the driver. The goal is to infer properties of the edges and drivers' preferences for edge properties under the assumption that drivers are taking shortest paths according to noisy copies of an underlying cost function. Given inferred driver preferences and edge costs, it is then possible to make predictions about the routes that will be taken by a driver on edges that have never been encountered by the driver before. For example, we can imagine learning from a driver traversing the streets of London and then make predictions about what routes the driver will prefer in Toronto. Alternatively, if city planners were considering changing road structures and they wanted to forecast how drivers would behave given a new road topology, a shortest path factorization model might be a good choice. In the factorization problem, we assume that driver-specific edge costs have a low-rank structure.

More specifically, paths are assumed to be shortest paths according to the driver's cost function. The cost function for a path is the sum of costs of edges on the path. Noise-free driver-specific edge costs are computed as the inner product of trait vector $\mathbf{U}_e \in \mathbb{R}^K$ for each edge e with a driver-specific preference vector $\mathbf{V}_d \in \mathbb{R}^K$ for each driver d. Noisy edge costs are drawn independently from Gaussian distributions with mean equal to the noise-free cost that are truncated to ensure that edge costs are non-negative.

The SPF problem is to infer $\mathbf{U}$ and $\mathbf{V}$ from the observations of paths. Intuitively, suppose that edges correspond to road segments, and drivers are members of the driving population. Paths are the routes that drivers take to get from home to work, from home to the grocery store, from a family member's house to the gas station, etc. The assumption is that there are a small number of traits that characterize each road segment. For example, real roads vary based on the average speed of traffic, start-stop frequency, the risk of traffic build-ups, their crowdedness, the scenery, the degree to which being an aggressive driver helps speed progress, etc. The degree to which a road segment e has such traits would be the kind of information stored in $\mathbf{U}_e$. The corresponding dimensions of $\mathbf{V}$ would then denote how important each of these traits is to each driver. Some drivers may be aggresive drivers concerned only about the total transit time, while others may prefer a minimal stress drive, even if it is slower. These different types of drivers could be represented via different $\mathbf{V}_d$ vectors. As in other matrix factorization-based algorithms like used in recommendation systems (Rennie and Srebro, 2005; Salakhutdinov and Mnih, 2007), it is not assumed that the traits are given ahead of time. The assumption is simply that this low rank structure exists, and it is up to the learning algorithm to discover which edges and drivers have which traits.

3.7 Shortest Path Factorization with RandOMs

This section describes how to apply the RandOM formulation to the SPF problem.

3.7.1 Generative Model

The RandOM generative model for SPF given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is as follows:

$$\mathbf{U}_e \sim \text{Gaussian}(\mathbf{0}, \sigma^2 \mathbf{I}) \qquad \text{for each } e \in \mathcal{E} \tag{3.13}$$

$$\mathbf{V}_d \sim \text{Gaussian}(\mathbf{0}, \sigma^2 \mathbf{I}) \qquad \text{for each } d = 1, \ldots, D \tag{3.14}$$

where σ^2 is a fixed variance.

Next sample each path conditional upon a driver d, a start node s, and an end node t. To sample each path:

$$w_e \sim \text{TruncGaussian}(\mathbf{U}_e^\top \mathbf{V}_d + b, 1) \qquad \text{for each } e \in \mathcal{E} \tag{3.15}$$

$$\boldsymbol{y} = \text{ShortestPath}(s, t, \mathcal{G}, \boldsymbol{w}) \tag{3.16}$$

where b is a fixed bias, TruncGaussian(μ, σ^2) is a Truncated Gaussian that is constrained to be greater than 0, and ShortestPath$(s, t, \mathcal{G}, \boldsymbol{w})$ returns the shortest path from s to t in $\mathcal{G}$ using edge costs given by $\boldsymbol{w}$.

3.7.2 Learning

The learning problem is to observe the data set $\mathcal{D}$ and infer parameters $\mathbf{U}$ and $\mathbf{V}$. Learning is done via MCEM.

The EM objective (Neal and Hinton, 1998) for a single data point $J(\mathbf{U}, \mathbf{V}, Q; \mathcal{D}_n)$ is

$$\mathbb{E}_{\hat{\boldsymbol{w}} \sim Q(\cdot)} \left[\log \left(p(\boldsymbol{y}_n \mid \boldsymbol{w}) p(\hat{\boldsymbol{w}} \mid \mathbf{U}, \mathbf{V}) p(\mathbf{U}) p(\mathbf{V}) \right) - \log Q(\hat{\boldsymbol{w}}) \right]. \tag{3.17}$$

The EM algorithm alternates between performing E (expectation) steps and M (maximization) steps. In the E step, $\mathbf{U}$ and $\mathbf{V}$ are held fixed and $Q(\cdot)$ is updated to optimize J. Here there is a separate Q_n for each n. The standard result is that optimal choice for $Q_n(\cdot)$ is to set it equal to the posterior distribution

$$p(\boldsymbol{w} \mid \boldsymbol{y}_n, d_n, \mathbf{U}, \mathbf{V}) \propto 1\{\boldsymbol{y}_n = F(\boldsymbol{w})\} p(\boldsymbol{w} \mid \mathbf{U}, \mathbf{V}, d_n) \tag{3.18}$$

$$= 1\{\boldsymbol{y}_n = F(\boldsymbol{w})\} \prod_{e \in \mathcal{E}} \text{TruncGaussian}(w_e; \mathbf{U}_e^\top \mathbf{V}_{d_n}, 1). \tag{3.19}$$

In the M step, all $Q_n(\cdot)$ are held fixed, and J is optimized with respect to $\mathbf{U}$ and $\mathbf{V}$. The objective including all n is

$$\mathbf{U}, \mathbf{V} = \underset{\mathbf{U}', \mathbf{V}'}{\operatorname{argmax}} \sum_{n=1}^{N} \mathbb{E}_{\hat{\boldsymbol{w}} \sim Q_n(\cdot)} \left[\log \left(p(\hat{\boldsymbol{w}} \mid \mathbf{U}', \mathbf{V}') p(\mathbf{U}') p(\mathbf{V}') \right) \right]. \tag{3.20}$$

These updates are not tractable to perform exactly, so instead an incremental MCEM algorithm is used (Neal and Hinton, 1998). In this variant, for each n, L samples $\hat{\boldsymbol{w}}^{(n1)}, \ldots, \hat{\boldsymbol{w}}^{(nL)}$ are drawn from (3.19) using a specialized slice sampler (described below). Then the M step objective is replaced with a Monte Carlo approximation:

$$\sum_{n=1}^{N} \frac{1}{L} \sum_{l=1}^{L} \left[\log \left(p(\hat{\boldsymbol{w}}^{(nl)} \mid \mathbf{U}, \mathbf{V}) p(\mathbf{U}) p(\mathbf{V}) \right) \right], \tag{3.21}$$

and $\mathbf{U}$ and $\mathbf{V}$ are updated using a small number of steps of gradient ascent.

3.7.3 Slice Sampling for the E Step

This section describes how to implement the Monte Carlo E step using a specialized slice sampler. The section begins by reviewing slice sampling, and then it describes how to combine slice sampling with combinatorial algorithms to obtain a fast sampler. This section describes a slice sampler tailored to the shortest paths problem, and it makes a general observation that may lead to minor improvements over Tarlow et al. (2012) for general RandOM slice samplers.

3.7.3.1 Review of Slice Sampling

Slice sampling (Neal, 2003) is a Markov Chain Monte Carlo (MCMC) method. It has favorable properties over alternatives like Metropolis Hastings in being less sensitive to parameters of a proposal distribution, and it has been shown to mix in polynomial time when run on log concave distributions (Lovász and Vempala, 2003). Tarlow et al. (2012) describe a specialization of slice sampling to RandOM models.

Slice sampling is used to draw samples from an unnormalized probability distribution $\tilde{p}(w)$. The basic idea is to sample uniformly from the region $R = \{(w, u) : 0 < u < \tilde{p}(w)\}$ using an MCMC algorithm that alternates between resampling w and resampling u. The definition of R ensures that $\int p(w, u) du \propto \tilde{p}(w)$, so it is valid to jointly sample w and u, and then discard the u components of the sample.

Slice sampling alternates between resampling w conditioned on u and u conditioned on w.

It works by repeatedly applying a transition kernel that leaves the distribution invariant. Starting from a current point w_0, the next point is chosen as follows:

- Sample $u \sim \text{Uniform}(0, \tilde{p}(w_0))$ (note: this should all be implemented in log-space).
- Sample w uniformly from the *slice*, $\{w' : \tilde{p}(w') > u\}$.

We say that w is *in the slice* if $\tilde{p}(w) > u$. The second step cannot always be implemented exactly, so Neal (2003) gives alternative updates that leave the uniform distribution over the slice invariant. The main suggestion is to do the following:

- Construct a random initial interval $[w_l, w_r]$ such that $w_0 \in [w_l, w_r]$.
- Step outwards by incrementing $w_l = w_l - \alpha$ until $\tilde{p}(w_l) < u$, where $\alpha \in \mathbb{R}_{>0}$ is a parameter that controls the speed at which the interval is expanded. Similarly, step outwards by incrementing $w_r = w_r + \alpha$ until $\tilde{p}(w_r) < u$. At this point, a contiguous section of the slice lies completely within the interval.
- Step inwards by sampling $\hat{w} \sim \text{Uniform}(w_l, w_r)$. If $\hat{w}$ is in the slice, then finish and transition to $\hat{w}$. Otherwise, shrink the interval so that w_0 remains inside the interval and $\hat{w}$ is one of the endpoints. Repeat the stepping inwards step.

The above describes how to use slice sampling with a 1D w. To handle higher dimensions as will be needed when sampling $\boldsymbol{w} \in \mathbb{R}^P$, a standard approach is to choose a random direction $\Delta \in \mathbb{R}^P$ uniformly from the surface of a sphere centered at $\boldsymbol{w}$, and then to sample along the line defined by $\boldsymbol{w} + \lambda\Delta$ for $\lambda \in (-\infty, \infty)$.

3.7.3.2 Specialization to General RandOMs

This section gives guidance on how slice samplers should be implemented in general for RandOM models.

The problem is to perform one step of slice sampling on the MCEM posterior (e.g., (3.19)). We are given an initial $\boldsymbol{w}_0 \in F^{-1}(\boldsymbol{y})$ and a direction Δ, and would like to choose a λ that leaves the distribution invariant (i.e., do a slice sampling update). The key idea is to define three sub-slices.

- The *legal slice* $\{\lambda : \mathcal{L}(\boldsymbol{w} + \lambda\Delta)\}$.
- The $\boldsymbol{y}$*-slice* $\{\lambda : \boldsymbol{w} + \lambda\Delta \in F^{-1}(\boldsymbol{y})\}$.
- The *prior slice* $\{\lambda : p(\boldsymbol{w} + \lambda\Delta \mid \mathbf{U}, \mathbf{V}, d) > u\}$.

The slice is then the intersection of these three sub-slices.

There are then properties of the subslices that can be useful to improve efficiency.

Convexity. The first source of efficiency is convexity, which can arise in all three types of sub-slice (but in any specific model may only arise in a subset of the sub-slices). For example:

1. If $\mathcal{L}(\boldsymbol{w})$ measures whether all dimensions of $\boldsymbol{w}$ are positive, then the legal slice is a convex set.
2. $F^{-1}(\boldsymbol{y})$ can be defined as $\{\boldsymbol{w} : f_{\boldsymbol{w}}(\boldsymbol{y}) \leq f_{\boldsymbol{w}}(\boldsymbol{y}') \quad \forall \boldsymbol{y}' \in \mathcal{Y}\}$. If $f_{\boldsymbol{w}}(\boldsymbol{y})$ is a CRF-like energy function as discussed in Section 3.3.3, then $f_{\boldsymbol{w}}(\boldsymbol{y})$ is a linear function of $\boldsymbol{w}$ (see (3.4)), so $F^{-1}(\boldsymbol{y})$ is an intersection of halfspaces and thus convex set, and the $\boldsymbol{y}$-slice is also a convex set.
3. If $p(\boldsymbol{w} \mid \ldots)$ is a log-concave distribution, then the prior slice is a convex set.

Convexity of the individual slices can be leveraged during the Stepping In phase of slice sampling. Since the initial point $\boldsymbol{w}_0$ will always be inside the slice and the interval resulting from the Stepping Out phase will always have endpoints outside the slice, convexity implies that there is a single transition point in between $\boldsymbol{w}_0$ and each endpoint where one leaves each convex sub-slice. For example, suppose for some $\hat{\lambda} > 0$, $\boldsymbol{w}_0 + \hat{\lambda}\Delta$ is in the $\boldsymbol{y}$-slice but not in the slice (maybe the point is not in the prior slice); it is then immediately known that $[0, \hat{\lambda}]$ is fully contained in the $\boldsymbol{y}$-slice, and there is no need for calling an expensive combinatorial algorithm for any later λ in this range that is encountered; it suffices to simply return `true`. When the union of the subslices is substantially different from the intersection, this can provide significant savings.

Combinatorial Algorithms for the $\boldsymbol{y}$-slice The second type of efficiency comes from the combinatorial optimization view of $F(\boldsymbol{w})$ and the fact that a slice sampling step always starts with a setting of $\boldsymbol{w}_0$ that is in the $\boldsymbol{y}$-slice.

This source of efficiency can be leveraged in addition to convexity structure if both are present. There are three ways of framing the problem of testing whether a particular $\boldsymbol{w} \in F^{-1}(\boldsymbol{y})$:

1. Run a combinatorial optimization algorithm with weights $\boldsymbol{w}$ and check whether $\boldsymbol{y}$ is an argmin.

2. Suppose we have recently solved a combinatorial optimization algorithm with weights $\boldsymbol{w}'$. Use a dynamic combinatorial optimization algorithm to update the solution to be the one for $\boldsymbol{w}$, and check whether $\boldsymbol{y}$ is an argmin.

3. Suppose we have recently solved a combinatorial optimization algorithm with weights $\boldsymbol{w}'$ and that $\boldsymbol{y}$ was an argmin. Check whether the argmin changes given weights $\boldsymbol{w}$.

Tarlow et al. (2012) shows how to use (2) to improve efficiency for bipartite matching RandOMs using dynamic combinatorial algorithms. The new observation here is that (3) can be more efficient than (2). Details for the shortest path case are given in the next section.

Ordering the Slices. The final suggestion is to test whether a point is in the slice by checking each of the sub-slices in order of least expensive to most expensive, and to short-circuit the computation as soon as a point is determined not to be in any of the sub-slices, since this implies that the point is not in the slice. This saves runs of the more expensive sub-slice computations and also makes the implementation more convenient by checking for legality of a point before calling the combinatorial optimization.

3.7.3.3 Efficiently Handling the y-slice with Shortest Path Trees

Given a source node s in a weighted graph $\mathcal{G}$ with all edge costs > 0, we can run Dijkstra's algorithm to get a *shortest path tree.* A shortest path tree is represented via a pointer from each node $v \neq s$ to a parent $pa(v)$, and a cost $c(v)$ for each node. Such a structure is a shortest path tree if $c(v)$ represents the distance from s to v via the shortest path in $\mathcal{G}$ and if the last step in the shortest path from s to v is to go from v's parent to v. This implies $c(v) = c(pa(v)) + w_{pa(v),v}$, where w_{uv} is the cost of edge uv.

An interesting property of shortest path trees is that they can be verified more efficiently than they can be constructed. They can be constructed in $\mathcal{O}(|\mathcal{V}| \log |\mathcal{V}| + |\mathcal{E}|)$ time using Dijkstra's algorithm but verified in $\mathcal{O}(|\mathcal{E}|)$ time using a simple loop over edges (Cormen et al., Exercise 24.3-4 Solution).

To leverage this property within the slice sampler, we need a fast method for proposing a shortest path tree $\mathcal{T}(\boldsymbol{w})$ given a shortest path tree $\mathcal{T}(\boldsymbol{w}_0)$.

The new suggestion is to keep the parent structure of $\mathcal{T}(\boldsymbol{w})$ fixed and update the node costs $c(v)$ so that $c(v) = c(pa(v)) + w_{pa(v),v}$. By iterating over nodes in topological order, this can be done in one loop over nodes ($\mathcal{O}(|\mathcal{V}|)$ time). We can then run the verification algorithm on the newly proposed shortest path tree. If the verification algorithm succeeds, then we have proven that $\boldsymbol{w}$ is in the $\boldsymbol{y}$-slice. If the verification algorithm fails, then it is necessary to run a more expensive check (e.g., run Dijkstra's algorithm from scratch), since it is possible for the structure of the shortest path tree to change while leaving the shortest path from s to some target node v unchanged. However, perhaps there is a more efficient method for determining whether the shortest path has changed; this could be studied in future work. In general, the suggestion when working with RandOMs is to focus on the dynamic combinatorial verification problem (which returns true or false as to whether the argmin has changed) instead of focusing on the dynamic combinatorial optimization problem (which returns a full configuration).

The verification procedure is most useful in the Stepping Out phase of slice sampling. If α is chosen to be small, then it will induce small changes in $\boldsymbol{w}$ that do not affect the structure of the shortest path tree. In these cases, the above procedure provides a fast way of verifying that a particular λ remains in the $\boldsymbol{y}$-slice.

3.8 Experiments

3.8.1 Baseline model

The goal in choosing a baseline model is to illustrate a common tradeoff when modelling structured data: models that ignore the combinatorial structure of the data can be appealing because they are often simpler to train, and sometimes a post-hoc cleanup step can enforce the combinatorial constraints (e.g., using rejection sampling). The baseline model adopts this philosophy.

The baseline model ignores the combinatorial structure of paths and produces a distribution that factorizes fully over the choice of each edge. More specifically, the approach follows 3-way factored models (Memisevic and Hinton, 2007; Krizhevsky et al., 2010; Kiros et al., 2014). There are three input components: the driver d, the start and end nodes s and t, and the edge identity e. Given the three inputs, the model produces a probability that edge e is used $p(u_e)$ in the shortest path from s to t. The goal of the model is to assign high probability to edges that are used on an observed path and low probability to edges that are not used. A training instance is then composed of the tuple (d, s, t, u_e^*).

More specifically, $p(u_e \mid d, s, t)$ is defined as

$$p(u_e \mid d, s, t) = \sigma \left(\mathbf{U}_e^\top \left(\mathbf{V}_d \oplus (\mathbf{T}_s + \mathbf{T}_t) \right) \right), \tag{3.22}$$

where $\sigma(\cdot)$ is the logistic sigmoid and $\oplus$ is either elementwise addition or multiplication for additive and multiplicative variants of the model, respectively. $\mathbf{U} \in \mathbb{R}^{|\mathcal{E}| \times K}$, $\mathbf{V} \in \mathbb{R}^{D \times K}$, and $\mathbf{T} \in \mathbb{R}^{|\mathcal{V}| \times K}$ are parameter matrices for edges, drivers, and nodes respectively. Subscripts select rows, so there is a K-dimensional real-valued representation vector for each entity. Note that the $\mathbf{T}$ parameters are needed so that the distribution over which edges are used is a function of the start and end points of the path. The training objective is then a standard maximum likelihood objective that can be optimized with gradient ascent.

3.8.2 Data

To test the RandOM model on the SPF problem we create a data set of N paths describing the routes of $D = 3$ drivers traversing a square grid graph with dimensions 3×6. We synthesise this data, by constructing $K = 2$ dimensional ground truth trait vectors $\mathbf{U}_{\text{gt}}$ and $\mathbf{V}_{\text{gt}}$ from which we generate noisy edge costs

$$w \sim \text{TruncGaussian}(\mathbf{U}_{\text{gt}}^\top \mathbf{V}_{\text{gt}}, \eta^2), \tag{3.23}$$

where η sets the scale of the noise. For simplicity, we start by setting the elements of the trait vectors to be random numbers uniformly drawn from $[0, 1)$. Later, we will consider a more carefully crafted $\mathbf{U}_{\text{gt}}$ designed to highlight differences between the baseline and RandOM models (see Section 3.8.4).

Using these edge costs we construct an element $(d_n, \boldsymbol{y}_n)$ in the data set by picking a random driver, $d_n \in \{1, 2, 3\}$, and random distinct nodes, s_n and t_n on the graph and then constructing the shortest path $\boldsymbol{y}_n$ from s_n to t_n according a sample from w_{e,d_n}.

3.8.3 Quantitative Results as a Function of Noise and Data Size

To measure the performance of the learned parameters $\mathbf{U}$ and $\mathbf{V}$, we draw 3×10^3 samples from the RandOM model to obtain Monte-Carlo estimates of $\log(p(\boldsymbol{y}_n \mid \mathbf{U}, \mathbf{V}, d_n))$ for each path $(d_n, \boldsymbol{y}_n)$ in the data. We report the "training score" as the average of these log probabilities over the training data and similarly compute a "test score" for 200 test paths not seen during training. If none of the Monte-Carlo samples match $\boldsymbol{y}_n$, we remove $\boldsymbol{y}_n$

from the evaluation procedure and separately report the proportion of such failures as the sampling failure rate.

For the baseline model, we compute the equivalent training and test scores using

$$p(\boldsymbol{y}_n \mid \mathbf{U}, \mathbf{V}, \mathbf{T}, d_n) = \prod_{e \in \boldsymbol{y}_n} p(u_e \mid d_n, s_n, t_n) \prod_{e \notin \boldsymbol{y}_n} (1 - p(u_e \mid d_n, s_n, t_n)) \,. \tag{3.24}$$

We find that the baseline model assigns a significant probability to configurations of edges which do not correspond to valid paths between s_n and t_n. A simple fix for this is to reject these samples at test time until a valid path is produced, but this comes at a computational cost. The score of this rejection-sampled baseline is analytically computed on our small 3×6 node example by enumerating all valid paths, $\mathcal{Y}(s_n, t_n)$, between s_n and t_n and then evaluating

$$\frac{1}{N} \sum_{n=1}^{N} \left[\log \left(p(\boldsymbol{y}_n \mid \mathbf{U}, \mathbf{V}, \mathbf{T}, d_n) \right) - \log(A_n) \right], \tag{3.25}$$

where

$$A_n = \sum_{\boldsymbol{y} \in \mathcal{Y}(s_n, t_n)} p(\boldsymbol{y} \mid \mathbf{U}, \mathbf{V}, \mathbf{T}, d_n). \tag{3.26}$$

The average value of A_n is the typical acceptance rate for the rejection sampler which gives an indication the computational inefficiency of this method.

Figure 3.1(a) shows the convergence of the training and test scores during the training of a RandOM model on a data set of $N = 100$ paths generated with noise $\eta = 0.01$. We find that even after the scores have plateaued, the values of $\mathbf{U}$ and $\mathbf{V}$ continue to evolve, indicating a flat objective function near the chosen solution. At convergence, the RandOM model considerably

Model	**Training Score**	**Test Score**	**Test Acceptance**
Baseline ($\oplus$: multiply)	-8.035	-8.655	1.0
+Rejection	-0.389	-0.572	0.003
Baseline ($\oplus$: add)	-7.594	-8.369	1.0
+Rejection	-0.337	-0.542	0.003
RandOM	-0.097(0%)	-0.337(3.5%)	1.0

Table 3.1: Quantitative results for data size 100 and noise 0.01. Numbers in parentheses indicate the sample failure rate.

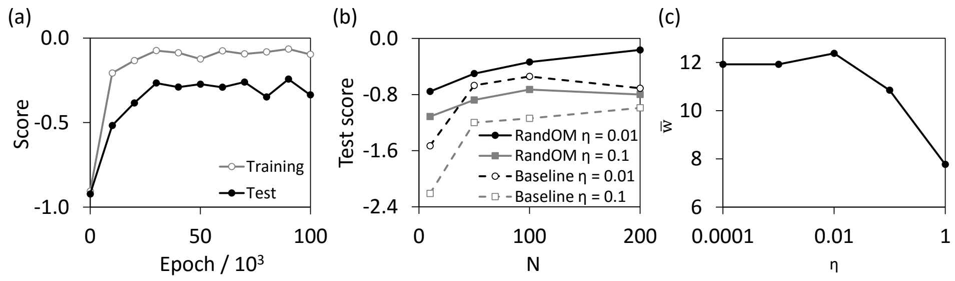

Figure 3.1: Performance of the RandOM model on the SPF problem. (a) Convergence of the training and test scores for a RandOM model trained on a data set with $N = 100$ and $\eta = 0.01$. (b) Comparison of the RandOM model with the baseline ($\oplus$: add) as a function of N for $\eta \in \{0.1, 0.01\}$. (c) The decay of the mean magnitude of the edge costs found by the RandOM model trained on $N = 100$ paths as the noise in the data set increases

outperforms the baseline models in predicting the shortest path taken by the drivers. This superiority remains true even with costly rejection sampling of the baseline model at test time (see table Table 3.1). We find that surprisingly few paths are required in the training data set for the RandOM model to achieve a good performance at test time (see Figure 3.1(b)), and for all parameters (N, η) we tested the RandOM model outperforms the baselines.

Besides inferring $\mathbf{U}$ and $\mathbf{V}$, we can also ask whether the RandOM model captures the noise in the training data. The RandOM model can represent variability in paths with a fixed standard deviation in (3.23) by changing the magnitude of $\boldsymbol{w}$; smaller (larger) values cause the fixed noise to have less (more) effect on which paths are chosen. Comparing (3.23) and (3.15), we expect the mean magnitude, $\bar{w}$, of the elements of $[\mathbf{U}^\top\mathbf{V}+b]$ to scale as η^{-1}. In Figure 3.1(c) we do not see this precise scaling, but we can correctly observe the decay of $\bar{w}$ with increasing η.

Here we have shown that the RandOM model quantitatively outperforms the baseline in a simple scenario. In the next section we describe a different scenario, which is engineered to highlight the key qualitative difference between the models.

3.8.4 Qualitative Results: Bias Resulting from Ignoring Combinatorial Structure

In the E step, the RandOM model only samples configurations of edge costs which are consistent with the shortest path structures observed in the data. The baseline model, in contrast, treats each edge independently,

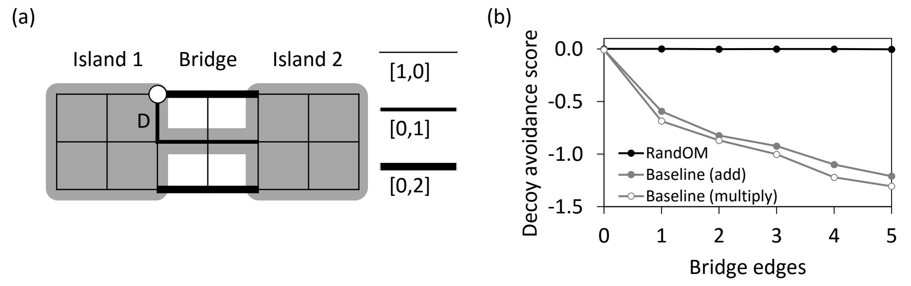

Figure 3.2: Biasing the baseline model. (a) By carefully arranging cheap, expensive and "impassable" edges (black lines), we implement a scenario resembling two islands linked by a bridge (grey outline). For the case illustrated the bridge contains 2 edges. We add the avoided decoy edge (D) and create a data set of paths starting at the circled node. (b) The score (representing the mean log probability oveer the test set of generating valid paths avoiding the decoy edge) for the RandOM model and baselines as a function of the bridge length.

and tries to learn to assign a high probability to edges used frequently in the training data (conditioning on the path start and end nodes). In this section we present an exaggerated scenario where the baseline's ignorance of the combinatorial structure in the data significantly hampers its performance.

We create a square grid graph that consists of three types of edge:

- "cheap" edges have feature vectors $[1, 0]$
- "expensive" edges have feature vectors $[0, 1]$
- "impassable" edges have feature vectors $[0, 2]$

The appropriate qualitative properties of these edges can be obtained by setting all driver feature vectors to $[v_n, 1]$, where $v_n \ll 1$. We build two separate "islands" of cheaply-linked nodes and connect these islands with impassable edges. Then we allow one path of expensive edges (a "busy bridge") to link the islands. Finally, we place a single "decoy" expensive edge on one of the islands which is never used in the ground truth paths due to it's cost (see Figure 3.2(a)). During training we give this carefully constructed $\mathbf{U}_{\text{gt}}$ to the models and only learn the remaining parameters.

If we observe drivers crossing from one island to the other, the baseline model will interpret the expensive edges on the bridge as being desirable, since they are used frequently. This bias means that the baseline will assign a significant probability for using the decoy edge even though this is inconsistent with the observed paths when correctly interpreting the constraints of the problem: if a driver is trying to get from one island to the other, there is no choice but to use the bridge, so the fact that the bridge is used should be irrelevant to determining the desirability of the decoy edge.

Instead, one should only look at whether the decoy edge is used or avoided, and in the data it is always avoided. The RandOM model correctly makes this inference and learns to avoid the decoy edge.

We generate a set of $N = 100$ training paths and separate set of 200 test paths on our engineered graph with edge cost noise $\eta = 0.01$. All paths start at one end of the decoy edge and finish at randomly chosen points on the graph.

Here we score the models by how often they produce samples which correctly avoid the decoy edge when trained on this data. For the RandOM model, the decoy avoidance score is computed as the average of Monte Carlo estimates of $\log\left(p(\boldsymbol{y} \in \mathcal{Y}_{\bar{D}}(s_n, t_n) \mid \mathbf{U}_{\text{gt}}, \mathbf{V}, d_n)\right)$ over the test set, where $\mathcal{Y}_{\bar{D}}(s_n, t_n)$ is the set of valid paths between s_n and t_n avoiding the decoy edge. For the baseline model, we again consider the case where invalid paths are rejected and compute the decoy avoidance score as

$$\frac{1}{N}\sum_{n}\left[\log\left(\sum_{\boldsymbol{y}\in\mathcal{Y}_{\bar{D}}(s_n,t_n)} p(\boldsymbol{y} \mid \mathbf{U}_{\text{gt}}, \mathbf{V}, \mathbf{T}, d_n)\right) - \log(A_n)\right]. \tag{3.27}$$

Figure 3.2(b) shows how these scores vary as we increase the length of the bridge between the islands. As the bridge extends there are more observations of drivers on expensive edges, which increasingly biases the baseline towards paths containing the decoy edge. In contrast, the RandOM model correctly interprets the shortest path structures in the data as indicating that the decoy edge is undesirable.

3.9 Related Work

There are several areas related to RandOMs. One place where there has been significant interest in perturbation-based models is in online learning, and in particular on Follow the Perturbed Leader algorithms (Kalai and Vempala, 2005). These algorithms have been applied to online learning in combinatorial settings such as shortest paths (Takimoto and Warmuth, 2003; Kalai and Vempala, 2005). See Chapter 8 for a detailed discussion of how perturbations are used and can be understood in the online learning setting.

For the purpose of semi-supervised learning, Blum et al. (2004) construct random graphs and find min-cuts that agree with labeled data. This leverages the idea of solving random combinatorial optimization problems, but no learning algorithm is presented. Perturb and MAP (P&M) (Papandreou and Yuille, 2011) learn structured models that involve a combinatorial optimization algorithm within the model definition, focusing on the case of using efficient minimum cut algorithms for image segmentation. The modelling formulation is very similar, although the RandOM formulation seems to extend more naturally to a broader range of models and optimization procedures. The main difference comes in the approach to learning. P&M proposes a moment-matching objective that is easy to optimize and that works well in practice, but the probabilistic underpinnings are less clear; i.e., learning is not directly maximizing the likelihood of observed data under the generative model. It is also not clear how, for example, P&M would be extended to a fully Bayesian treatment. Hazan and Jaakkola (2012) develops an understanding of how the expected score of the argmax configuration relates to the partition function of the more traditional Gibbs distribution. Gane et al. (2014) delves deeper into the correlation structure that results from using perturbation models with factorized perturbations.

There are other approaches to learning probabilistic structured prediction models to optimize high order utility functions. As mentioned previously, Gane et al. (2014) propose an improved Hard EM algorithm for the RandOM formulation that avoids a degeneracy that is heuristically worked around by Tarlow et al. (2012). Kim et al. (2015) employ an empirical risk minimization approach that directly minimizes expected losses in RandOM-like models using the combinatorial structure of the optimizer in order to do more efficient integration. Premachandran et al. (2014) propose a pragmatic approach of producing a set of diverse M-best proposals with combinatorial optimization algorithms (Batra et al., 2012), and then re-calibrating a probabilistic model over the proposals for use within a Bayesian decision theory-like decision procedure. The downside of this approach is that it is a two-stage procedure without a single objective function to optimize. For the shortest paths application, Ratliff et al. (2006) present a max-margin based approach that leverages efficient search procedure; however, there is no probabilistic interpretation.

A somewhat different line of work that shares the basic motivation is variational autoencoders (Kingma and Welling, 2014), generative adversarial networks (Goodfellow et al., 2014), and generative moment matching networks (Li et al., 2015). The generative adversarial networks and moment matching networks use different learning objectives from maximum likelihood. The commonality is that a generative model is built around highly efficient deterministic primitives; in these cases, rather than using a combinatorial optimization algorithm, these works use neural networks as the primitive. More precisely, if we let $\boldsymbol{w} = (\theta, \boldsymbol{u})$, where θ are neural network parameters and $\boldsymbol{u}$ is random noise, then we could define $F(\boldsymbol{w})$ to be the result of applying a neural network parameterized by θ to inputs $\boldsymbol{u}$. To make most sense in this analogy, the output should be a structured discrete object, such as a sentence. This formulation would apply equally if θ were a parameter or a random quantity as in Bayesian formulations of neural networks. The challenge with this direction is that in the RandOM formulation, $F^{-1}(\boldsymbol{y})$ is typically more structured than such a neural net formulation, which makes the sampling in the E step more plausibly effective. It is not immediately obvious, for example, how one would find a $\boldsymbol{w} = (\theta, \boldsymbol{u})$ such that $F(\boldsymbol{w}) = \boldsymbol{y}$ for a given a $\boldsymbol{y}$, much less sample from the space of such $\boldsymbol{w}$'s. However, if this could be done effectively then an MCEM algorithm analogous to the RandOM formulation would be a reasonable learning formulation.

3.10 Discussion

This chapter reviewed Randomized Optimum Models (RandOMs) and presented a new application of RandOMs to the problem of factorizing shortest paths into edge-specific and driver-specific trait vectors. The key computational challenge in RandOM formulations is developing a sampler for the E step of Monte Carlo EM. For this problem, slice sampling is particularly well-suited, and this chapter gives an additional illustration beyond Tarlow et al. (2012) about how to construct a slice sampler that takes advantage of the combinatorial structure in the problem. While it may be appealing to design simpler models that ignore the combinatorial structure present in the data (such as the baseline from Section 3.8.1), it is shown in Section 3.8.4 that this can lead to biases in the learned model that cause the wrong qualitative conclusions to be drawn from the observed data.

Looking forward, we would like to apply a similar formulation to models of highly structured natural data such as images and text, and to explore optimization routines beyond standard combinatorial optimization algorithms.

3.11 References

D. Batra, P. Yadollahpour, A. Guzman-Rivera, and G. Shakhnarovich. Diverse m-best solutions in markov random fields. In *Computer Vision–ECCV 2012*, pages 1–16. Springer, 2012.

A. Blum, J. Lafferty, M. R. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the twenty-first international conference on Machine learning*, page 13. ACM, 2004.

A. Bouchard-Côté and M. I. Jordan. Variational inference over combinatorial spaces. In *Advances in Neural Information Processing Systems*, pages 280–288, 2010.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

A. Gane, T. Hazan, and T. Jaakkola. Learning with maximum a-posteriori perturbation models. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 247–256, 2014.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

T. Hazan and T. Jaakkola. On the partition function and random maximum a-posteriori perturbations. *arXiv preprint arXiv:1206.6410*, 2012.

A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

A. Kim, K. Jung, Y. Lim, D. Tarlow, and P. Kohli. Minimizing expected losses in perturbation models with multidimensional parametric min-cuts. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, 2015.

D. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

R. Kiros, R. Salakhutdinov, and R. Zemel. Multimodal neural language models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 595–603, 2014.

D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. 2009.

A. Krizhevsky, G. E. Hinton, et al. Factored 3-way restricted boltzmann machines for modeling natural images. In *International Conference on Artificial Intelligence and Statistics*, pages 621–628, 2010.

Y. Li, K. Swersky, and R. Zemel. Generative moment matching networks. *arXiv preprint arXiv:1502.02761*, 2015.

L. Lovász and S. Vempala. Hit-and-run is fast and fun. 2003.

J. Lubin. A human vision system model for objective image fidelity and target detectability measurements. In *Proc. EUSIPCO*, volume 98, pages 1069–1072, 1998.

R. Memisevic and G. Hinton. Unsupervised learning of image transformations. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

V. Movahedi and J. H. Elder. Design and perceptual validation of performance measures for salient object segmentation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 49–56. IEEE, 2010.

R. M. Neal. Slice sampling. *Annals of Statistics*, 31(3):705–767, 2003.

R. M. Neal and G. E. Hinton. A new view of the EM algorithm that justifies incremental and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands, 1998.

A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Migrating code with statistical machine translation. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 544–547. ACM, 2014.

S. Nowozin. Optimal decisions from probabilistic models: the intersection-over-union case. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 548–555. IEEE, 2014.

G. Papandreou and A. Yuille. Perturb-and-MAP random fields: Using discrete optimization to learn and sample from energy models. In *Proceedings of the IEEE International Conference on Computer Vision*, 2011.

P. Pletscher and P. Kohli. Learning low-order models for enforcing high-order statistics. In *AISTATS*, 2012.

V. Premachandran, D. Tarlow, and D. Batra. Empirical minimum bayes risk prediction: How to extract an extra few% performance from vision models with just three more parameters. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1043–1050. IEEE, 2014.

N. Ratliff, J. A. Bagnell, and M. Zinkevich. Maximum margin planning. In *International Conference on Machine Learning*, 2006.

J. D. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

C. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. *The Journal of Machine Learning Research*, 4:773–818, 2003.

D. Tarlow and R. Zemel. Structured output learning with high order loss functions. In *Artificial Intelligence and Statistics (AISTATS)*, 2012.

D. Tarlow, R. P. Adams, and R. S. Zemel. Randomized optimum models for structured prediction. In *Artificial Intelligence and Statistics (AISTATS)*, 2012.

B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*, volume 16, page 25. MIT Press, 2004.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, 2005.

L. G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

G. C. Wei and M. A. Tanner. A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. *Journal of the American statistical Association*, 85(411):699–704, 1990.