Computing Expected Losses in Perturbation Model using Multidimensional Parametric Min-cuts

Adrian Kim **Kyomin Jung**
Seoul National University
{adkim955,kjung}@snu.ac.kr

Daniel Tarlow **Pushmeet Kohli**
Microsoft Research
{dtarlow,pkohli}@microsoft.com

Abstract

We consider the problem of computing and differentiating expected losses under perturbation-based probabilistic models. This is a challenging computational problem that has traditionally been tackled using Monte Carlo-based methods. In this work, we show how a generalization of parametric min-cuts can be used to address the same problem, achieving high accuracy faster than a sampling-based baseline.

1 Introduction

Many problems in machine learning can be formulated as structured-output prediction, such as pixel labelling problems in computer vision and protein side-chain prediction in bio-informatics. A key challenge in the solution of these problems is to build structured prediction models that capture key correlations within the outputs and to learn these models from data. There are a range of approaches to this problem, including training a deterministic predictor to minimize (regularized) empirical risk (e.g., structural SVMs [1, 2]), PAC Bayesian-based approaches where the goal is to train a randomized predictor to minimize a regularized empirical risk [3], and probabilistic modelling paired with Bayesian decision theory [4].

Perturbation models [5, 6, 7] are an approach that have been a focus of interest in recent years, and are closely related to both PAC-Bayesian approaches and probabilistic modelling. The idea is to build a probabilistic model over structured outputs by drawing a random energy function and then returning the argmin of the random energy function as a sample from the model. These models can then be trained under maximum likelihood-like objectives [5, 6, 7] or to minimize expected loss [3, 8]. Typically the distribution over energy functions is restricted so that the optimization step is tractable (e.g., it is a min-cut problem). When this is the case, perturbation models have the desirable property that exact samples can be drawn efficiently with a single call to an efficient optimization procedure.

Our aim in this work is to revisit the problem of training perturbation models to minimize expected losses. Previous works [3, 8] have used Monte Carlo-based methods to estimate the needed gradients. A concern with these approaches is that the gradient estimates can have high variance, as is the case with the well-known REINFORCE algorithm [9]. Instead, our approach here is to explore combinatorial methods that take advantage of the structure of the optimization problem in order to more efficiently make use of optimizer runs. As a first foray into this approach, we restrict attention to the case where the perturbation model takes the form of a uniform distribution over model parameters followed by a call to a min-cut/maxflow routine.

Our method is based on a recently proposed generalization [10] of the parametric min-cut algorithm [11] which in the 1-dimensional case is able to efficiently compute all parameter values (breakpoints) at the which the minimum energy (MAP) solution changes. To demonstrate the efficacy of our method, we compare estimated expected losses and their gradients computed by our method with those obtained from a sampling-based scheme. Experimental results show that we get more

accurate solutions with fewer calls to the optimization procedure and less overall wall time. We conclude the paper by discussing ideas for generalizing this work to more general settings.

2 Background: Perturbations, Expected Losses

We will focus on the case where perturbation models are used to define a conditional probability model $P(y \mid x; \theta)$, where x is an input (e.g., an image), $y \in \{0,1\}^n$ is a structured output (e.g., a foreground-background image segmentation), and $\theta \in \mathbb{R}^m$ is a real-valued vector of parameter values. We additionally assume access to a feature vector $\phi(x, y) \in \mathbb{R}^m$ which contains unary and pairwise potentials. Perturbation models begin by defining an energy function $E(y \mid x; \theta) = \theta^T \phi(x, y)$. The second component to a perturbation model is the noise distribution $P(\gamma)$ which is a distribution over noise vectors $\gamma \in \mathbb{R}^m$. The probability model $P(y \mid x; \theta)$ can then be defined as follows:

$$\gamma \sim P(\gamma) \tag{1}$$

$$y = \operatorname*{argmin}_{y'} E(y' \mid x; \theta + \gamma). \tag{2}$$

It will be useful to define *minimizer* $f(\theta) = \operatorname{argmin}_y E(y \mid x; \theta)$, *dual function* $g(\theta) = \min_y E(y \mid x; \theta)$, and *inverse set* $f^{-1}(y) = \{\theta : f(\theta) = y\}$. Under this definition, the probability of a configuration y can be expressed as $P(y \mid x; \theta) = \int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) d\gamma$. We are interested in expected losses under perturbation models. The expected loss (or *risk*) is a function of a given y^* (in our case, the ground truth configuration) and parameters θ. It is defined as $R(y^*, \theta) = \sum_{y \in \{0,1\}^n} P(y \mid x; \theta) L(y^*, y) = \sum_{y \in \{0,1\}^n} \int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) L(y^*, y) d\gamma$ where $L(y^*, y)$ assigns a loss value for predicting y when the ground truth is y^*. The ultimate goal we are working towards is to learn parameters θ so as to minimize $R(y^*, \theta)$. In this work, as a first step, we focus on the prerequisite tasks of computing and differentiating $R(y^*, \theta)$.

3 Algorithm Description

We begin by making some assumptions. First, let $P(\gamma)$ be a uniform distribution such that $\theta + \gamma$ is distributed uniformly over a m-dimensional hyperrectangular region $S = \prod_{i=1}^m [w_i^-, w_i^+]$. Also assume that the minimizer $f(\theta)$ is unique for all θ except for a set with measure zero, so $f(\theta)$ can be treated as having a unique value. Finally, assume that for all $\theta \in S$, $E(y \mid x; \theta)$ is submodular and can be optimized efficiently.

In the following, it will be convenient for us to redefine $f^{-1}(y)$ so that only regions in S are included. That is, $f^{-1}(y) = \{\theta : f(\theta) = y \wedge \theta \in S\}$. Then from above, we have that $R(y^*, \theta) = \sum_{y \in \{0,1\}^n} \int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) L(y^*, y) d\gamma$. Noting that $L(y^*, y)$ is not a function of γ and that $\int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) d\gamma = \operatorname{Volume}(f^{-1}(y)) / \operatorname{Volume}(S)$, we can rewrite $R(\cdot)$ as $\sum_{y \in Y_S} L(y^*, y) \operatorname{Volume}(f^{-1}(y)) / \operatorname{Volume}(S)$, where $Y_S = \{y : \exists \theta . \theta \in S \wedge f(\theta) = y\}$ is the set of configurations that are minimizers for some $\theta \in S$.

In this paper, we introduce a novel method to find the minimizers $y \in Y_S$ and their inverse sets by iteratively updating a graph structure that we call a *skeleton*. Note that for a fixed y, $E(y \mid x; \theta)$ is a linear function of θ, which implies that the dual function $g(\theta) = \min_y E(y \mid x; \theta)$ is a piecewise concave function, where pieces are hyperplanes corresponding to minimizers y. Let h_y be the corresponding hyperplane for some fixed minimizer y. Intuitively, the skeleton $G_Y = (V_Y, E_Y)$ is a graphical representation of the dual $g(\theta)$ over S. The skeleton will be constructed on the given parameter space S by finding new minimizers, or hyperplanes, at each iteration until there are no more minimizers. At each iteration, the growing skeleton represents part of the dual $g(\theta)$ which is defined by the minimizers $y \in Y \subseteq Y_S$ found by the algorithm.

Definition 1 (Partial dual g_Y)**.** *For some given minimizer set $Y \subseteq Y_S$, let $g_Y(\theta) = \min_{y \in Y} E(y \mid x; \theta)$ be the partial dual, which is a piecewise concave function.*

For some given partial dual $g_Y(\cdot)$, each hyperplane h_y has a corresponding graph which we refer as a *facet* G_y. A facet $G_y = (V_y, E_y)$ is defined as the smallest convex hull made by the intersections of h_y and other hyperplanes, where V_y, E_y are boundary vertices and edges of the convex hull. Let

θ_v be the parameter value and $z_v = g_Y(\theta_v)$ corresponding to the vertex v. Note that a facet can be cut because of the boundaries the given parameter space makes. A skeleton is defined using the union of these facets as follows.

Definition 2 (Skeleton of g_Y over S)**.** *For some given partial dual g_Y, the skeleton of g_Y on S can be represented by the following structure $G_Y = (V_Y, E_Y)$. Let (u, v) be an edge between u and v, where $u, v \in V_Y$.*

- $V_Y = \bigcup_{y \in Y} \{v : \textit{Boundary vertices of } G_y, \textit{ where } \theta_v \in S, z_v = g_Y(\theta_v)\}$
- $E_Y = \bigcup_{y \in Y} \{(u, v) : \textit{Boundary edges of } G_y, \textit{ where } u, v \in V_Y\} \cup \{(u, v) : u \in V_Y, \theta_u \in \Pi_{i=0}^{m} \{w_i^-, w_i^+\}, v = (\theta_u, -\infty)\}$

For example, Figure 1b is a skeleton over some parameter space $S \in \mathbb{R}^2$ given a partial dual g_Y, where $Y = \{y_1, ..., y_5\}$. There are five facets on the skeleton, where four are cut by the boundaries of S.

From the given definitions, it is clear that an inverse set $f_Y^{-1}(y) = \{\theta : y = \text{argmin}_{y' \in Y} f_\theta(y') \wedge \theta \in S\}$ of y defined on a partial dual g_Y directly corresponds to the projection of the facet G_y. Thus, in order to calculate the volume of $\theta_Y(y)$, we can use the projected vertices of G_y on S. One of the main points of our method is that we are able to track every facet with every iteration, so that we can calculate the approximate expected loss every time we update the skeleton.

We now describe our *Skeleton method* and how it works. Figure 1 describes a visual example on how a skeleton is constructed and updated by a single iteration of our algorithm. Table 1 of the Appendix gives the pseudo code of the algorithm.

Initialization. The initial skeleton $G_Y = (V_Y, E_Y)$ is given by the following.

- $Y = \phi$
- $V_Y = \{(\theta_{v_n}, z_{v_n}) : \theta_{v_n} = \Pi_{i=0}^{m} \{w_i^-, w_i^+\}, z_{v_n} = \infty\}$
- $E_Y = \{(u, v) : u \in V_Y, v = (\theta_u, -\infty)\}$

Finding a New Facet. In order to find a new facet, the algorithm first picks some vertex $u = (\theta_u, z_u) \in V_Y$. Using the given oracle, a new solution $y_u = f(\theta_u)$ can be found. The first step is to determine if the new solution improves the current dual in any region. This can be determined by checking if $h_{y_u}(\theta_u) < z_u$. If this is the case, we say that a *cut* is made, and y_u is added to Y.

Next, we must find the new intersection points where h_{y_u} intersects other hyperplanes defining the partial dual. The key property of the new intersection points is that they will either appear at existing vertices $v \in V_Y$, or they appear on an edge $(p_h, p_t) \in E_Y$ that "crosses" the new hyperplane; that is $h_{y_{p_h}}(\theta_{p_h}) < z_{p_h}$ and $h_{y_{p_t}}(\theta_{p_t}) > z_{p_t}$. The set of vertices where $h_{y_v}(\theta_v) < z_v$, form a connected component in G_Y, and the crossing edges are the boundary edges of this connected component. Thus, the intersection points can be found by exploring a search tree outwards from u. When a node v is encountered such that $h_{y_v}(\theta_v) < z_v$, the intersection point between v and its parent is computed via a simple calculation, and the search tree is not searched further down that path. Upon termination, nodes within the connected component notated as H, are removed from V_Y, and the new intersection points notated as I are added. A step of this procedure is illustrated in Figure 1b, where there is a cut after selecting vertex u_i, colored in red.

Updating the Skeleton G_Y. When a cut is done in the skeleton, it should be updated with the new lower bound made by h_{y_u}. The nontrivial case is when some intersection point $p \in I$ is a new point made on some edge $(p_h, p_t) \in E_Y$, which is (u_1, v_1) for p_1 in Figure 1b. The new vertex p is added to all facets which share the edge (p_h, p_t). Also, a new edge (p_t, p) should be added to the skeleton. Boundary edges made from the convex hull of the new polytope G_{y_u} are also added to E_Y. Finally, the skeleton update is done when all head vertices $r \in H$ are deleted from every facet and all edges including r are removed from E_Y.

Calculating Expected Loss. At this point, the skeleton is fully updated. To compute expected losses, we use an off-the-shelf subroutine for computing the volume of each inverse set $f^{-1}(y)$ for $y \in Y$. The volumes are multiplied by the loss value for each y, and the products are summed to get the full expected loss. For normalization, the value is divided by the volume of S.

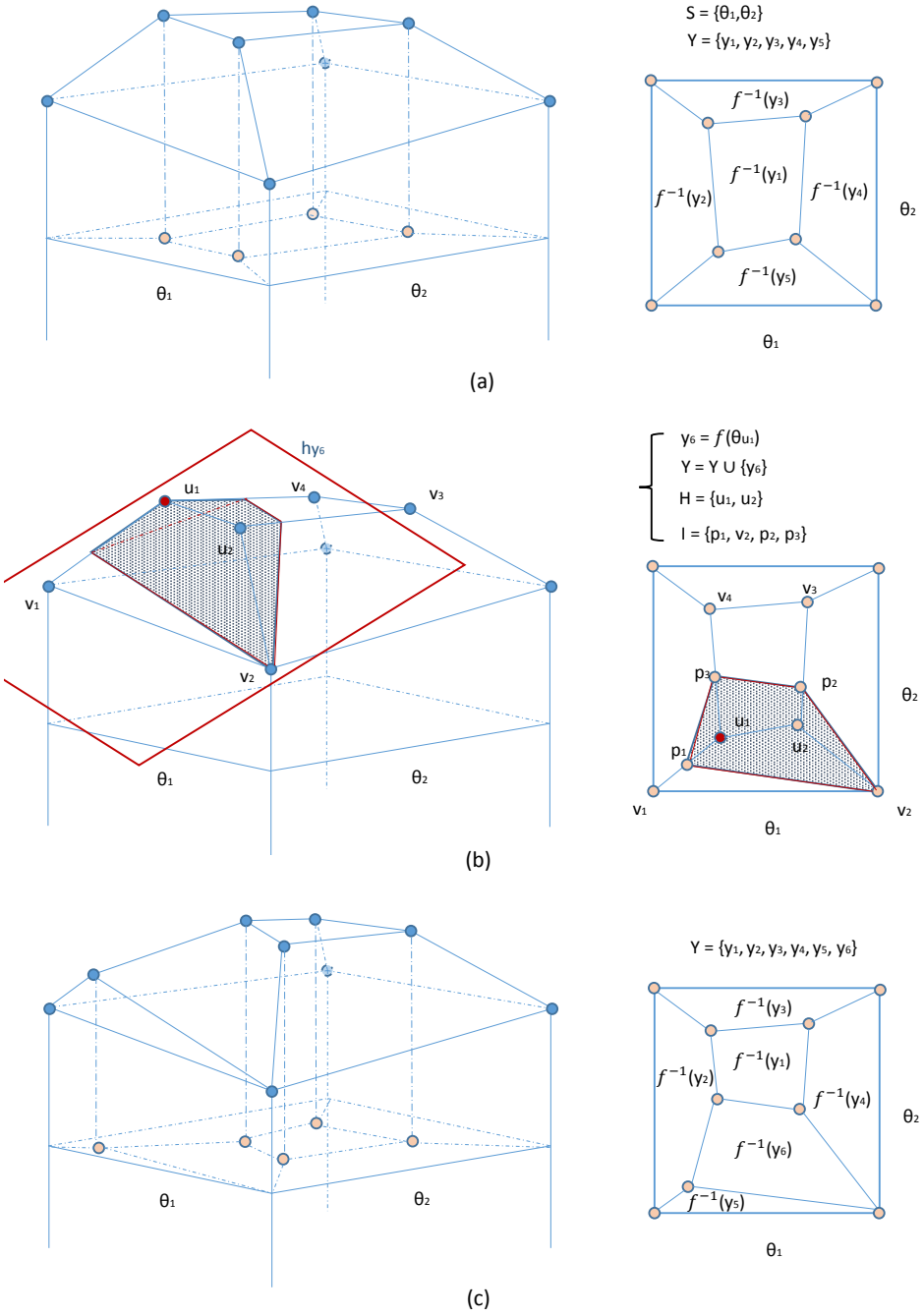

Figure 1: A visual example of the skeleton method with two parameters. Notice that the left side represents the partial dual g_Y and that the right image is the projection of facets on the given parameter space $S = \Pi_i^2[w_i^-, w_i^+]$. We use notate the parameter set for y as f_y for convenience. (a) The skeleton after iterating five times. There are five facets on the parameter space so that the expected loss is $R(y^*, \theta) = \sum_{n=1}^{5} \text{Volume}(f^{-1}(y_n))L(y^*, y_n)/\text{Volume}(S)$ (b) Suppose we take some unused vertex u_1. In this case, we can see that the hyperplane h_{y_6} makes a cut in the skeleton. (c) By updating the skeleton, a new facet f_6 is found. Since there is a unique loss value for the facet, we can calculate the expected loss as $R(y^*, \theta) = \sum_{n=1}^{6} \text{Volume}(f^{-1}(y_n))L(y^*, y_n)/\text{Volume}(S)$.

4 Experiments

Data and Setup. In this section, we apply the Skeleton Method to a foreground-background image segmentation task, comparing against a Monte Carlo baseline which estimates expected loss by drawing samples from the prior and reporting the average incurred loss. The energy function is of the following form:

$$E(y \mid x; \theta) = \theta^T \phi(x, y) = E(x) + \theta_1 \sum_{i}^{n} (y_i \neq x_i) + \theta_2 \sum_{(y_i, y_j) \in E_v} (y_i \neq y_j) + \theta_3 \sum_{(y_i, y_j) \in E_h} (y_i \neq y_j),$$

where E_v, E_h are each sets of neighboring vertical and horizontal pairs of pixels respectively. The x_i's in the unary potentials are noisy versions of the ground truth, where background pixels are flipped to foreground with uniform probability of 5%. Images are of size 90x120 ($n = 9600$ pixels). Ground truth segmentations for the images used are shown in Figure 2g-2i.

We calculated the expected loss for a uniform noise distribution $P(\gamma)$ which defines a parameter space $S \subseteq \mathbb{R}^m$. Each parameter has certain ranges, or *windows* which can be selected by $P(\gamma)$. We used 3 windows $\theta_1 = [-1, 1], \theta_2 = [0, 1], \theta_3 = [0, 1]$ for the experiments. The loss function for the experiments was defined as the Hamming distance, $L_H(y^*, y) = \sum_{i=1}^{n} (y_i \neq y_i^*)$, although we note that the formulation supports arbitrary loss functions.

Performance of the Skeleton Method. To evaluate the methods, it would be ideal to have a ground truth value. Unfortunately this is hard to calculate accurately, because the Skeleton method does not always run to termination in the time that we give it, and there is necessarily some variance in the estimates returned by the sampling estimate. Thus, we report the estimates from each method along with 95% confidence intervals derived from the sampling method. For the sampling method, in each trial, parameters were independently sampled 1 million times, and this was repeated 10 times.

Figure 2 shows plots of expected losses calculated by the two methods in runtime. The average sampling estimate (across all trials) appear as red dashed lines with 95% confidence ranges for the Monte Carlo estimator in Figure 2a-2c. Also shown are the cumulative averages for three representative trials of the sampling (green to blue curves), and the Skeleton method (magenta). The main take-away is that the expected loss values of both methods converge to similar values, but particularly with few samples, there is high variance in sampling. While the Monte Carlo estimator has significant variance even after 1000 seconds, the *Skeleton Method* has essentially converged to its final, accurate estimate after 10 seconds. This suggests that we can stop running the method in the middle of the algorithm to estimate the expected loss with high accuracy. The reason such behavior appears is related to the high concentration of vertices in the later iterations of the algorithm. Many calculations made in later iterations induce inverse sets which have very small volumes, implying the low contribution to the expected loss.

Gradients. Looking forward to the learning task, we next turn attention to calculating gradients of the expected loss with respect to θ. As an initial approach, we use a finite differences approximation, $\frac{R(y^*,(\theta_1,\theta_2,\theta_3+\delta))-R(y^*,(\theta_1,\theta_2,\theta_3))}{\delta}$, where $\delta = 0.01$. Assuming that the Monte Carlo estimator gives us a close estimate to the true value after many samples, we can find the expected gradient by taking the differences of expected losses from two parameter spaces S, S', where $S' = S + (0, 0, \delta)$. For the gradient estimation, we used the true value as the average estimated gradient value from 10 trials of sampling 100k times. Comparing with this, we want to estimate the gradient with the Skeleton Method by the following approach. We take thin *slices* $s_+ = \Pi_i^2[w_i^-, w_i^+] \times [w_3^+, w_3^+ + \delta]$ and $s_- = \Pi_i^2[w_i^-, w_i^+] \times [w_3^-, w_3^- + \delta]$ which are actually the spaces added and deleted when S shifts to S'. To find the derivative, we take the expected losses from each slice and get the differences. Results are shown in Figure 2d-2f. We see that we can find a very stable value of the gradient by this method and compute quickly compared to the sampling method.

5 Discussion

Preliminary results show that the Skeleton method is a promising alternative to Monte Carlo methods in the example problems we considered. The primary challenge going forward is to broaden the applicability of the method, extending to higher dimensions and enlarging the space of supported perturbation distributions. It is likely in these cases that exactness of the method will need to be

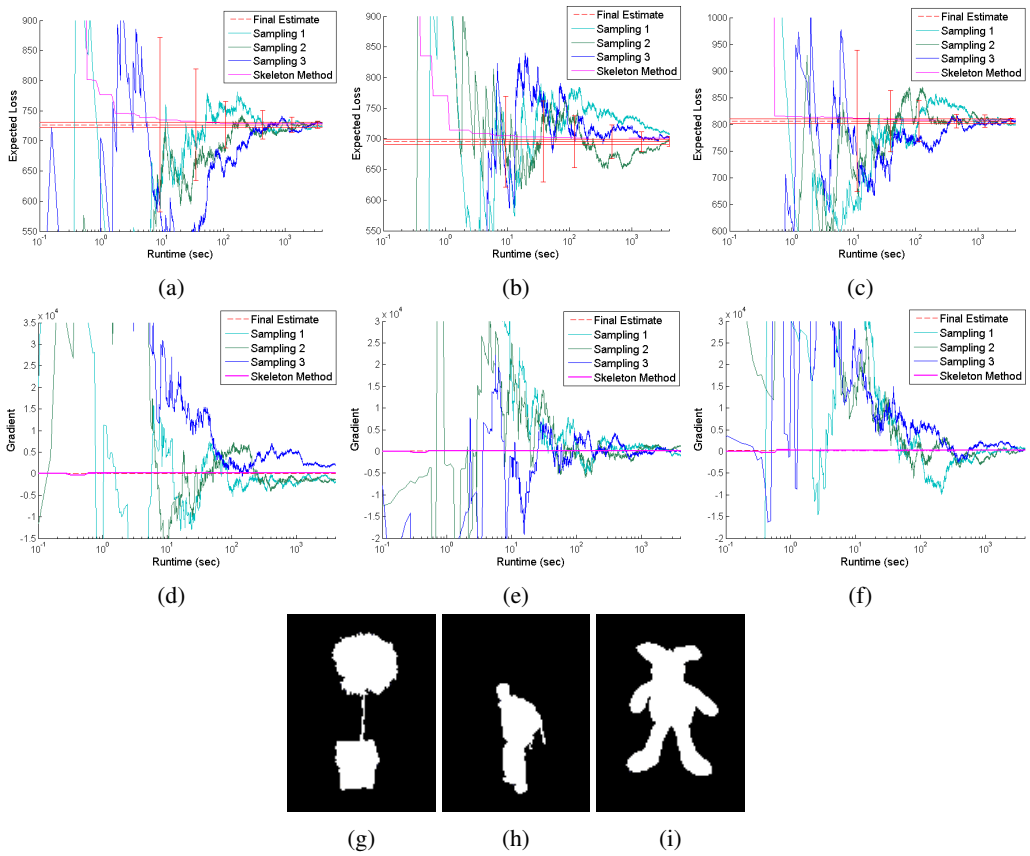

Figure 2: (a)-(c): Comparison of expected losses between the Monte Carlo estimator and the Skeleton Method with three images by runtime. (d)-(f): Comparison of gradients calculated by both methods where $\delta = 0.01$.(g)-(i): Images used in experiments. Images are taken from [12].

abandoned due to the fact that the number of solutions will likely grow, and the computations of necessary volumes will become computationally hard. Despite this, we believe the algorithm presented here will be useful going forward. There are two possibilities we are interested in exploring: first, using a hybrid of the Skeleton and sampling methods where some dimensions are sampled and some are integrated analytically using the Skeleton method (producing a Rao-Blackwellized sampler); second, we believe there to be opportunities for computing and differentiating upper bounds based on the Skeleton structure, which could lead to interesting new learning methods.

References

[1] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems*, page None, 2003.

[2] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005.

[3] Joseph Keshet, David McAllester, and Tamir Hazan. Pac-bayesian approach for minimization of phoneme error rate. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2224–2227. IEEE, 2011.

[4] Uwe Schmidt, Qi Gao, and Stefan Roth. A generative perspective on mrfs in low-level vision. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1751–1758. IEEE, 2010.

[5] G. Papandreou and A. Yuille. Perturb-and-MAP Random Fields: Using Discrete Optimization to Learn and Sample from Energy Models. In *ICCV*, pages 193–200, Barcelona, Spain, November 2011.

[6] Daniel Tarlow, Ryan Prescott Adams, and Richard S Zemel. Randomized Optimum Models for Structured Prediction. In *AISTATS*, pages 21–23, 2012.

[7] Tamir Hazan and Tommi S Jaakkola. On the Partition Function and Random Maximum A-Posteriori Perturbations. In *ICML*, pages 991–998, 2012.

[8] Tamir Hazan, Subhransu Maji, Joseph Keshet, and Tommi Jaakkola. Learning efficient random maximum a-posteriori predictors with non-decomposable loss functions. In *Advances in Neural Information Processing Systems*, pages 1887–1895, 2013.

[9] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[10] Yongsub Lim, Kyomin Jung, and Pushmeet Kohli. Multi-dimensional parametric mincuts for constrained map inference. *arXiv preprint arXiv:1307.7793*, 2013.

[11] Giorgio Gallo, Michael D Grigoriadis, and Robert E Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[12] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 309–314. ACM, 2004.

Appendix 1 : Pseudo code of Skeleton method

Require: Oracle f
Ensure: Expected Loss R
  $(Y, G_Y) \leftarrow InitSkeleton()$
  for all $u = (\theta_u, z_u) \in V_i$ **do**
    $y_u = f(\theta_u)$
    if $h_{y_u}(\theta_u) < z_u$ **then**
      Add y_u to Y
      $(I, H) \leftarrow FindIntersection(G_Y, h_{y_u})$
      Add $f_{y_u} = (y_u, I)$ to F_Y
      $V_Y = (V_Y \cup I) - H$
      for all Intersection vertices $p \in I$ **do**
        if p is a new vertex **then**
          Add p to all $G_y \in \{$Facets sharing (p_t, p_h), where p_h is above and p_t is below $h_{y_u}\}$
          Append new edge (p_t, p) to E_Y
        end if
      end for
      Remove head vertices $r \in H$ from all facets
      Remove $E_- = \{(u, v) : u$ or $v \in H\}$ from E_Y
      Append $E_+ = \{$Boundary edges of $G_{y_u}\}$ to E_Y
      $R = \sum_{y \in Y} Volume_{f^{-1}(y)} L(y^*, y)$
    end if
  end for
  return R

Table 1: $SkeletonMethod()$