Distributed Algorithms for Large Scale Learning and Inference in Graphical Models

Alexander G. Schwing, Tamir Hazan, Marc Pollefeys, *Fellow, IEEE* and Raquel Urtasun, *Member, IEEE*

Abstract—Over the past few years we have witnessed an increasing popularity in the use of graphical models for applications in computational biology, computer vision and natural language processing. Despite the large body of work, most existing learning and inference algorithms can neither cope with large scale problems which require huge amounts of memory and computation nor can they effectively parallelize structured learning with a small number of training instances. In this paper, we propose a family of model-parallel learning and inference algorithms which exploit distributed computation and memory storage to handle large scale graphical models as well as a small number of training instances. Our algorithms combine primal-dual methods and dual decomposition to preserve the same theoretical guarantees (and for certain parameter choices the same solution) as if they were run on a single machine. We demonstrate the capabilities of our algorithms in a wide variety of scenarios, showing the advantages and disadvantages of model-parallelization.

Index Terms—graphical models, big data, sample parallelism, model parallelism

1 Introduction

Most real-world applications are structured, *i.e.*, they are composed of multiple random variables which are related. For example, in natural language processing, we might be interested in parsing sentences syntacticly. In computer vision, we might want to predict the depth of each pixel, or its semantic category. In computational biology, given a sequence of proteins (*e.g.*, lethal and edema factors, protective antigen) we might want to predict the 3D docking of the anthrax toxin. While individual variables could be considered independently, it has been demonstrated that taking dependencies into account improves prediction performance significantly.

Prediction in structured models is typically performed by maximizing a scoring function over the space of all possible outcomes, an NP-hard task for general graphical models. Notable exceptions are graphical models with sub-modular energies or low tree-width structure where inference is performed exactly in polynomial time. Unfortunately, most real-world problems do not belong to this category and only approximate solutions can be obtained.

A wide variety of approaches have been proposed to obtain approximate solutions to the general inference problem, *e.g.*, via sampling [1] or via iterative algorithms that solve a max-flow problem [2]. While aforementioned approaches generally consider the problem more globally, message passing approaches are interesting due to the local structure. While effective in the case of small inference problems, most approaches assume that the graphical model and the involved variables fit into memory. This is an issue as the size of real-world graphical models is growing rapidly when operating with “big data.” These days, cameras for example have several mega-pixels, even in low-cost cellphones. Holistic approaches on the other hand aim at solving multiple related tasks, hence the complexity also grows with the number of jointly considered problems. But the challenge of large-scale models is not only related to memory consumption but also to computational complexity. Typical message passing algorithms for example scale with the number of dimensions involved in the largest function, *e.g.*, they are already inherently quadratic for graphical models involving pairwise connections.

Learning with these models is typically done in the context of regularized risk minimization, where a log-linear model is employed, and a convex surrogate loss

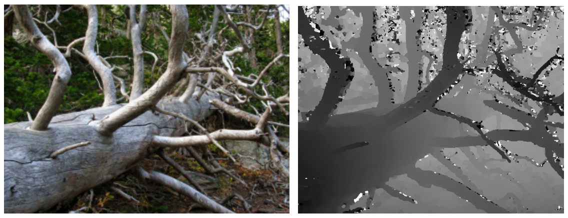
Fig. 1. 283 label disparity map computed from a 12 MPixel stereo pair.

- *A. G. Schwing and R. Urtasun are with the Computer Science Department, University of Toronto, Canada. E-mail: see http://alexander-schwing.de*
- *T.Hazan and M. Pollefeys are with University of Haifa, Israel and ETH Zurich, Switzerland.*

is optimized. Most learning algorithms have the same limitations as inference techniques, since they perform inference for each parameter update step. On the one hand, max-margin Markov networks (M^{3}Ns) or structured SVMs (SSVMs) [3], [4] minimize the structured hinge-loss, and perform a loss-augmented inference using cutting plane optimization. Conditional random fields (CRFs) [5] on the other hand, minimize the log-loss, and thus require the computation of the normalizer of a probability distribution, *i.e.*, the partition function, during each iteration. Primal-dual methods, such as [6], [7], have the advantage of interleaving learning and inference. As a consequence they do not require convergence of the inference task in order to update the weights. While this results in reduced computational complexity, they assume, just like CRFs, SSVMs and M^{3}Ns, that the graphical model and the messages can be stored on a single computer. Moreover training with mini-batches containing a small number of training instances is increasingly popular these days. However, the predominant data-parallel scheme is not very effective for this setting. This again limits the applicability of learning algorithms for large-scale problems and/or small mini-batch size.

To overcome these issues, in this work we propose a family of learning and inference algorithms which exploit distributed computation and memory storage to handle large scale graphical models like the one involved in the disparity computation illustrated in Fig. 1. Our algorithms combine primal-dual methods and dual decomposition to preserve the same theoretical guarantees (and for certain choices of parameters the same solution) as if they were run on a single machine. For the learning task we make use of interleaving previously suggested in [6], [7]. All in all the presented work extends [8], [9] to region graphs and provides a unified implementation. We demonstrate the capabilities of our algorithms in a wide variety of scenarios, showing the advantages and disadvantages of model-parallelization.

In the remainder, we first review both non-distributed inference and non-distributed learning algorithms. Afterwards we discuss a dual decomposition approach for parallelization of both task before describing some features of our publicly available implementation. We then illustrate the applicability of the proposed approaches via experiments before discussing related work.

2 Review: Inference and Learning

In this section we review inference and learning algorithms that employ message passing techniques. We first define the general setting.

Let $x \in \mathcal{X}$ be the input data, *e.g.*, images or sentences, and $s \in \mathcal{S}$ the output elements, *e.g.*, image segmentations or parse trees. We assume the output space elements s to lie within a discrete product space, *i.e.*, $s = (s_1, s_2, \ldots) \in \mathcal{S} = \prod_i \mathcal{S}_i$ with $\mathcal{S}_i = \{1, \ldots, |\mathcal{S}_i|\}$. Let $\phi : \mathcal{X} \times \mathcal{S} \to \mathbb{R}^F$ be a feature mapping which projects from the object and output product space to an F-dimensional feature space.

During *inference*, we are interested in finding the most likely output $s^* \in \mathcal{S}$ given some data $x \in \mathcal{X}$. To this end we maximize a distribution $\tilde{p}$ ranging over all output symbols. We model $\tilde{p}$ to be log-linear in some weights $w \in \mathbb{R}^F$, *i.e.*,

$$\tilde{p}(s \mid x, w) \propto \exp\left(w^\top \phi(x, s)/\epsilon\right). \tag{1}$$

We introduced the temperature ϵ to adjust the smoothness of the distribution, *e.g.*, for $\epsilon = 0$ the distribution is concentrated on the maximizers of the score $w^\top \phi(x, s)$, often also referred to as 'negative energy.'

The *learning* task is concerned with finding the "best" parameters w, given sample pairs (x, s) containing data symbol x and its corresponding groundtruth output s. In the following we briefly review both, inference and learning in greater detail, and provide the intuitions necessary to follow the derivations of the distributed algorithms.

2.1 Inference

During inference, we are interested in finding the most likely output space configuration $s^* \in \mathcal{S}$ given data $x \in \mathcal{X}$. More formally we aim at solving

$$s^* = \arg\max_{s \in \mathcal{S}} w^\top \phi(x, s). \tag{2}$$

This task is generally intractable, *i.e.*, NP-hard, given the exponentially many configurations within $\mathcal{S}$.

To overcome this issue, we follow a variational approach and approximate the distribution over outcomes with a simpler factorized distribution. Towards this goal, we rephrase the task as the one of finding a distribution $\tilde{p}(s)$ which minimizes the KL-divergence $D_{\mathrm{KL}}(\tilde{p} \mid \exp(w^\top \phi(x, s)/\epsilon))$ between the modeled distribution $\tilde{p}(s \mid x, w)$ given in Eq. (1) and $\tilde{p}(s)$. The corresponding program is equivalent to

$$\max_{\tilde{p} \in \Delta} \sum_{s \in \mathcal{S}} \tilde{p}(s)\theta(s \mid x) + \epsilon H(\tilde{p}), \tag{3}$$

with score $\theta(s \mid x) = w^\top \phi(x, s)$, the entropy $H(\tilde{p}) = -\sum_s \tilde{p}(s) \ln \tilde{p}(s)$ and Δ the probability simplex of corresponding dimension. Note that the task given in Eq. (3) is identical to the program given in Eq. (2) when $\epsilon = 0$ since the distribution $\tilde{p} \in \Delta$ will be any distribution over the set of maximizers $\arg\max_{s \in \mathcal{S}} \theta(s \mid x)$. However, we introduce smoothness via $\epsilon > 0$ which admits optimality guarantees of approximations that are easier to achieve [10], [11].

The exponentially sized output spaces render the optimization of even the smooth objective intractable in general, and assumptions as well as approximations are required. For many applications a single measurement ϕ_k, $k \in \{1, \ldots, F\}$, *i.e.*, an element of the feature vector ϕ, consists of functions that depend

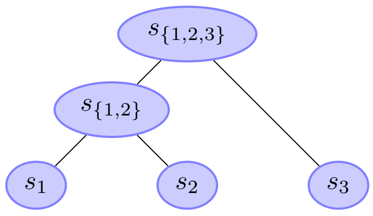

Fig. 2. Hasse diagram of a distribution that cannot be visualized by a factor graph. Note the sets of regions with $\mathcal{R} = \{\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,2,3\}\}$.

only on a fraction of the output variables. We refer to a subset of output space variables s_r using the index set r. Formally, $s_r = (s_i)_{i\in r}$ is a restriction of an output space object s to a subset of its variables. We assume the k-th measurement to be computable via

$$\phi_k(x, s) = \sum_{r\in\mathcal{R}_k} \phi_{k,r}(x, s_r),$$

where the set $\mathcal{R}_k$ subsumes all the index sets, subsequently also referred to as regions, required to compute the feature. Note that theoretically this is not a strong assumption since $\mathcal{R}_k$ may contain an element that refers to all variables.

Taking this decomposition into account we obtain

$$\theta(s \mid x) = \sum_{r\in\mathcal{R}, k: r\in\mathcal{R}_k} w_k \phi_{k,r}(x, s_r) = \sum_{r\in\mathcal{R}} \theta_r(s_r \mid x),$$

with the set of all unique regions being denoted $\mathcal{R} = \bigcup_k \mathcal{R}_k$, and the potential for region r referred to via

$$\theta_r(s_r \mid x) = \sum_{k: r\in\mathcal{R}_k} w_k \phi_{k,r}(x, s_r).$$

For clarity of the presentation we subsequently neglect the dependence of the potential on the input data, *i.e.*, we abbreviate $\theta_r(s_r \mid x)$ with $\theta_r(s_r)$.

Employing the aforementioned regions r, inference as defined in Eq. (3) is equivalently expressed as

$$\max_{\tilde{p}\in\Delta, \tilde{p}_r} \sum_{r,s_r} \tilde{p}_r(s_r)\theta_r(s_r) + \epsilon H(\tilde{p}) \text{ s.t. } \tilde{p}_r(s_r) = \sum_{s\setminus s_r} \tilde{p}(s),$$

with $\tilde{p}_r(s_r)$ being the true marginals arising from the distribution $\tilde{p}(s)$. This program remains intractable because of the exponentially sized summations incurring for computation of the entropy and the constraints. Typically we make use of fractional entropies [12] and approximate $H(\tilde{p})$ via a sum of tractable, local entropies while introducing counting numbers c_r as hyper-parameters, *i.e.*, $H(\tilde{p}) \approx \sum_r c_r H(\tilde{p}_r)$. In addition we relax the constraint space such that the true marginals $\tilde{p}_r(s_r)$ are no longer required to be globally consistent. To highlight this change from the marginal polytope to a local polytope, we use local beliefs $b_r(s_r)$ to refer to marginals that are no longer required to arise from a single joint distribution $\tilde{p}(s)$. The regions that we require to be consistent upon convergence are given by a parent-child relationship, *i.e.*, all the parents $P(r)$ of a region r are required to marginalize to the local belief b_r, *i.e.*, $\sum_{s_p\setminus s_r} b_p(s_p) = b_r(s_r)$ has to hold $\forall r, s_r, p \in P(r)$. Note that the set of parents $P(r)$ denotes a subset of those regions that strictly contain all elements of the set r. In addition to the set of parents we refer to the set of children of a region via $C(r) = \{c : P(c) = r\}$. Depicting regions as nodes in a graph, this parent-child relationship generalizes factor graphs. This is illustrated in Fig. 2 via a *Hasse diagram* for a simple approximation that cannot be depicted by common factor graphs.

Taking the decomposition assumption as well as the entropy and marginal polytope approximation into account, our resulting inference task reads as

$$\begin{array}{lll} \max\limits_{b} & \sum\limits_{r\in\mathcal{R}, s_r} b_r(s_r)\theta_r(s_r) + \sum\limits_r \epsilon c_r H(b_r) & \\ \text{s.t.} & \forall r & b_r \in \Delta \\ & \forall r, s_r, p \in P(r) & \sum_{s_p\setminus s_r} b_p(s_p) = b_r(s_r). \end{array} \quad (4)$$

Note that the local beliefs b_r are generally not constrained to arise from a joint distribution over all the involved variables. Enforcing such a global constraint is however possible at the expense of exponential complexity, by including a region that covers all the variables. It is generally sufficient to only include regions of size up to the tree-width of the problem when interested in finding the globally optimal solution, which corresponds to employing the junction tree algorithm. Hence the junction tree algorithm is part of the algorithms presented within this work.

Importantly, possible convergence guarantees depend on the users choice of counting numbers c_r. Letting $c_r \geq 0$ ensures concavity of the approximated primal program given in Eq. (4), and convexity of its corresponding dual to be illustrated below. In this case convergence to a global optimum can be guaranteed if care is taken regarding the method of optimization. This is known as *convex belief propagation* [13], [14], [15], [16]. Another choice for the counting numbers, not necessarily positive, is commonly referred to as *loopy belief propagation* [17] or *generalized loopy belief propagation* [18]. In this case $c_r = 1 - \sum_{a\in A(r)} c_a$ where $A(r)$ is the set of ancestors of region r. Note the difference between the set of parents $P(r)$ and the set of ancestors $A(r)$, *e.g.*, $P(\{1\}) = \{\{1,2\}\}$, while $A(\{1\}) = \{\{1,2\}, \{1,2,3\}\}$ for the example illustrated in Fig. 2. Due to neither concavity nor convexity of the primal loopy belief propagation program given in Eq. (4), convergence to the global optimum cannot be guaranteed for general graphs.

To leverage the structure of the program encoded within the constraints, more specifically the sparse parent-child relationship, it is beneficial to consider the dual problem if counting numbers admit. In any other case we can follow the steps to obtain the mes-

```
Algorithm: Message Passing Inference
Repeat until convergence
Iterate over $r$:
    $\forall p \in P(r), s_r \qquad \mu_{p\to r}(s_r) = \epsilon c_p \ln \sum_{s_p\setminus s_r} \exp \frac{\theta_p(s_p) - \sum_{p'\in P(p)} \lambda_{p\to p'}(s_p) + \sum_{r'\in C(p)\setminus r} \lambda_{r'\to p}(s_{r'})}{\epsilon c_p}$
    $\forall p \in P(r), s_r \qquad \lambda_{r\to p}(s_r) \propto \frac{c_p}{c_r + \sum_{p\in P(r)} c_p} \left(\theta_r(s_r) + \sum_{c\in C(r)} \lambda_{c\to r}(s_c) + \sum_{p\in P(r)} \mu_{p\to r}(s_r)\right) - \mu_{p\to r}(s_r)$
```

Fig. 3. A block-coordinate descent algorithm for the inference task.

$$\min_w \sum_{(x,s),r} \epsilon c_r \ln \sum_{\hat{s}_r} \exp\left(\frac{\tilde{\phi}_{(x,s),r}(x,\hat{s}_r) - \sum_{p\in P(r)} \lambda_{(x,s),r\to p}(\hat{s}_r) + \sum_{c\in C(r)} \lambda_{(x,s),c\to r}(\hat{s}_c)}{\epsilon c_r}\right) - w^\top d + \frac{C}{2}\|w\|_2^2$$

Fig. 4. The approximated learning task.

sage passing updates while noting that convergence is not guaranteed and typical primal-dual relations do not hold. The cost function of the dual is obtained by considering the Lagrangian of the program given in Eq. (4). To this end we introduce Lagrange multipliers $\lambda_{r\to p}(s_r)$ for the marginalization constraints of the beliefs, *i.e.*, we introduce a variable corresponding to the constraint $\sum_{s_p\setminus s_r} b_p(s_p) = b_r(s_r)$ for every region r, every element s_r of its domain and all its parents $p \in P(r)$. Taking into account the relationships encoded within the Hasse diagram we obtain the Lagrangian

$$\sum_r \left(\sum_{s_r} b_r(s_r)\left(\theta_r(s_r) - \sum_{p\in P(r)}\lambda_{r\to p}(s_r) + \sum_{c\in C(r)}\lambda_{c\to r}(s_c)\right) + \epsilon c_r H(b_r)\right).$$

Maximizing the Lagrangian w.r.t. $b_r(s_r)$ independently for reach restriction r using the primal-dual relationship between the logarithm of the parition function and the entropy (*cf*., [19]) yields the dual objective

$$\sum_r \epsilon c_r \ln \sum_{s_r} \exp \frac{\theta_r(s_r) - \sum_{p\in P(r)}\lambda_{r\to p}(s_r) + \sum_{c\in C(r)}\lambda_{c\to r}(s_c)}{\epsilon c_r},$$

which we aim at minimizing w.r.t. the Lagrange multipliers λ. In order to optimize this objective, we follow [16], [20] to derive a block-coordinate descent algorithm, where a region r is chosen iteratively, and the Lagrange multipliers $\lambda_{r\to p}(s_r)\ \forall p \in P(r), s_r$ are updated via a minimization computable in closed form. For completeness we summarize the update rules in Fig. 3.

Given the Lagrange multipliers λ, we recover the beliefs

$$b_r(s_r) \propto \exp \frac{\theta_r(s_r) - \sum_{p\in P(r)}\lambda_{r\to p}(s_r) + \sum_{c\in C(r)}\lambda_{c\to r}(s_c)}{\epsilon c_r}$$

if $\epsilon c_r > 0$. For $\epsilon c_r = 0$ the beliefs nonzero domain is determined by the maximizing elements

$$S_r^* = \arg\max_{s_r}\left(\theta_r(s_r) - \sum_{p\in P(r)}\lambda_{r\to p}(s_r) + \sum_{c\in C(r)}\lambda_{c\to r}(s_c)\right) \tag{5}$$

(*cf*. Danskin's theorem in, *e.g.*, [21]).

2.2 Learning

To derive common learning algorithms, let $\mathcal{D} = \{(x^{(i)}, s^{(i)})_{i=1}^N\}$ be the training set composed of data-output pairs, and let $\ell(s,\hat{s})$ be a task loss, which compares any output space configuration $\hat{s} \in \mathcal{S}$ with the ground truth labeling s, and returns a large number for estimates that are very different. Note that generally the task loss can be sample dependent, but we subsequently neglect this dependence for clarity of the exposition, *i.e.*, we let $\ell(s,\hat{s}) = \ell_{(x,s)}(s,\hat{s})\ \forall (x,s)$.

We follow [5], [3], [4] and frame learning as computing the maximum likelihood or max-margin estimator of the data. Let the loss-augmented likelihood of an estimate $\hat{s}$ given the data sample (x,s) be

$$q_{(x,s)}(\hat{s} \mid w) \propto \exp\left(\frac{w^\top \phi(x,\hat{s}) + \ell(s,\hat{s})}{\epsilon}\right).$$

We use the data-groundtruth pair (x,s) as an index rather than conditioning in order to avoid a cluttered notation. The negative log-likelihood of a data set with independently and identically distributed elements is expressed as $-\ln\left(\tilde{p}(w)\prod_{(x,s)\in\mathcal{D}} q_{(x,s)}(s \mid w)\right)$. Plugging in definitions for the prior $\tilde{p}(w)$ and the loss-augmented data likelihood $q_{(x,s)}(s \mid w)$, the cost function equals

$$\frac{C}{2}\|w\|_2^2 + \sum_{(x,s)\in\mathcal{D}} \epsilon \ln \sum_{\hat{s}\in\mathcal{S}} \exp\left(\frac{w^\top \phi(x,\hat{s}) + \ell(s,\hat{s})}{\epsilon}\right) - w^\top d \tag{6}$$

if we let $\tilde{p}(w) \propto \exp\left(\frac{-C}{2\epsilon}\|w\|_2^2\right)$, while the vector $d = \sum_{(x,s)\in\mathcal{D}} \phi(x,s)$ refers to the sum of empirical

Algorithm: Convex Structured Prediction

```
Repeat until convergence
    1) Iterate over all samples $(x,s)$ and regions $r$:
```

$$\forall p \in P(r), s_r \qquad \mu_{(x,s),p\to r}(s_r) = \epsilon c_p \ln \sum_{s_p\setminus s_r} \exp \frac{\tilde{\phi}_{(x,s),p}(s_p) - \sum\limits_{p'\in P(p)} \lambda_{(x,s),p\to p'}(s_p) + \sum\limits_{r'\in C(p)\setminus r} \lambda_{(x,s),r'\to p}(s_{r'})}{\epsilon c_p}$$

$$\forall p \in P(r), s_r \qquad \lambda_{(x,s),r\to p}(s_r) \propto \frac{c_p}{c_r + \sum\limits_{p\in P(r)} c_p}\left(\tilde{\phi}_{(x,s),r}(s_r) + \sum_{c\in C(r)} \lambda_{(x,s),c\to r}(s_c) + \sum_{p\in P(r)} \mu_{(x,s),p\to r}(s_r)\right) - \mu_{(x,s),p\to r}(s_r)$$

```
    2) Weight vector update with stepsize $\alpha$ determined to ensure the Armijo rule
```

$$w \leftarrow w - \alpha\left(\sum_{(x,s),r}\left(\sum_{\hat{s}_r} b_{(x,s),r}(\hat{s}_r)\phi_{(x,s),r}(\hat{s}_r) - \phi_{(x,s),r}(s_r)\right) + Cw\right)$$

Fig. 5. A block-coordinate descent algorithm for the learning task.

measurements. The second term of the cost function given in Eq. (6) arises from the partition function of the loss-augmented data log-likelihood $q_{(x,s)}(s \mid w)$. Moreover we assumed the loss of the groundtruth to equal zero, *i.e.*, $\ell(s,s) \equiv 0$. This term is commonly referred to as a soft-max function which smoothly approximates the max-function for $\epsilon \to 0$. For $\epsilon = 0$, the cost function given in Eq. (6) reads as

$$\frac{C}{2}\|w\|_2^2 + \sum_{(x,s)\in\mathcal{D}} \max_{\hat{s}\in\mathcal{S}} \left(w^\top\phi(x,\hat{s}) + \ell(s,\hat{s})\right) - w^\top d, \quad (7)$$

which linearly penalizes all configurations $\hat{s}$ for which $\max_{\hat{s}\in\mathcal{S}}\left(w^\top\phi(x,\hat{s}) + \ell(s,\hat{s})\right) \geq w^\top\phi(x,s)$, *i.e.*, the optimization penalizes outputs with score larger than the ground truth configuration. Note the similarity to M^{3}Ns [3] and SSVMs [4].

To our advantage, Eq. (6) is convex in w. However, the difficulty arises from the fact that we need to consider an exponentially sized output space $\mathcal{S}$. Minimizing the above cost function is therefore intractable in general and we reside to approximations.

In short, we can transform the primal learning program given in Eq. (6) to the dual domain. Due to the primal-dual relationship between the logarithm of the partition function and the entropy we obtain an intractable entropy ranging over the entire domain of the output space. To approximate we employ the decomposition assumption outlined when reviewing inference. In addition we use again the fractional entropy concept as well as local beliefs. Transforming the approximated dual back to the primal domain yields the program illustrated in Fig. 4. We let $\tilde{\phi}_{(x,s),r}(\hat{s}_r) = \sum_{k:r\in\mathcal{R}_k} w_k\phi_{(x,s),k,r}(x,\hat{s}_r) + \ell_{(x,s),r}(\hat{s}_r,s_r)$, while assuming – just like for inference – that the k-th feature vector element for every sample decomposes into a sum of regions $\mathcal{R}_k$, *i.e.*, $\phi_{(x,s),k} = \sum_{r\in\mathcal{R}_k}\phi_{(x,s),k,r}(x,\hat{s}_r)$.

Comparing the original learning task given in Eq. (6) with the approximation provided in Fig. 4, we observe that counting numbers c_r approximate the joint soft-max function, *i.e.*, the logarithm of the partition function, via a sum of local soft-max functions ranging over small subsets of output space variables s_r. To enforce consistency between the individual subsets of a sample (x,s), messages $\lambda_{(x,s),r\to p}(s_r)$ exchange information between a region r and its parents $P(r)$.

Note that the cost function given in Fig. 4 is convex when $\epsilon, c_r \geq 0$ and convergence is guaranteed when employing a block-coordinate descent approach which iterates between updating the weights w and the Lagrange multipliers λ. Even more importantly and to increase efficiency, convexity permits to interleave updates for the Lagrange multipliers λ and gradient steps w.r.t. the parameters w [6], [7]. It is useful to compute the gradient w.r.t. w without requiring convergence of the message passing.

Following [6], the resulting convex structured prediction algorithm is summarized in Fig. 5. While the first step gives the block-coordinate descent steps for the Lagrange multipliers λ, the second part illustrates the gradient update using beliefs

$$b_{(x,s),r}(s_r) \propto \exp\left(\left(\tilde{\phi}_{(x,s),r}(x,\hat{s}_r) - \sum_{p\in P(r)}\lambda_{(x,s),r\to p}(\hat{s}_r) + \right.\right.$$
$$\left.\left. + \sum_{c\in C(r)} \lambda_{(x,s),c\to r}(\hat{s}_c)\right)/(\epsilon c_r)\right)$$

if $\epsilon c_r > 0$. Following the inference schema, the beliefs nonzero domain is given by the set of maximizing states for $\epsilon c_r = 0$, *i.e.*, analogous to Eq. (5). We want to emphasize the similarity between the algorithm for message passing inference given in Fig. 3 and the first step within the convex structured prediction algorithm summarized in Fig. 5.

3 Distributed Inference

The learning and inference algorithms described in the previous section do not scale well to large prob-

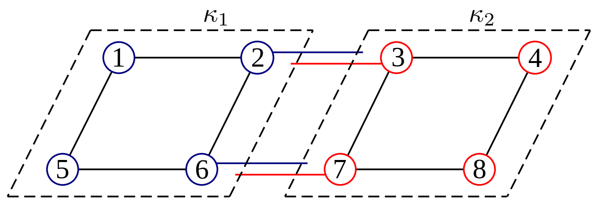

Fig. 6. The distributed architecture: partitioning the 8 variables onto two computers κ_1 and κ_2.

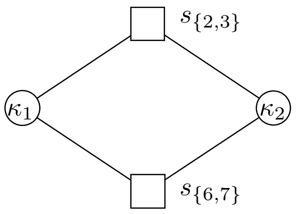

Fig. 7. The junction graph of the example in Fig. 6 for sending consistency messages ν between computers.

lems, as they require to store both the graphical model and the messages in memory. Moreover parallelization w.r.t. samples during learning is not very effective given only few examples. Furthermore, solving the approximate inference problem is computationally demanding, as it scales with the number of dimensions involved in the largest region, *i.e.*, they are already inherently quadratic for graphical models involving pairwise connections. This is a tremendous limitation for a wide variety of applications, where one is faced with very large graphical models with millions of nodes and many examples.

If we consider *shared memory* environments, *e.g.*, current standard multi-core computers, one strategy is application of a graph coloring of the Hasse diagram by making sure that regions within the Markov blanket are assigned different colors. Messages of all nodes having the same color are then computable in parallel. Considering large graphs, we can minimize the number of idle cores while directly optimizing the considered problem. Note that for densely coupled graphical models this strategy results in a large number of colors with few regions per color. The reader may keep in mind that graph coloring can potentially reduce convergence speed of inference since information might not propagate as quickly.

Computer clusters are however a cheaper alternative for large-scale computation when memory is the main limiting factor. In this setting, one can leverage the computational resources by distributing the problem onto all the available machines. Since we deal with *distributed memory* that is connected via local area networks, relatively slow compared to direct memory accesses, we cannot neglect the latency for transmitting messages between different computers. Therefore, we want to minimize the communication between computers while maintaining convergence guarantees particularly for strictly convex problems (*e.g.*, strictly positive counting numbers).

Towards this goal, we partition the output space $\mathcal{S}$. Note that this induces a partitioning of the region graph. We highlight that the likelihood of a region being assigned to multiple computers increases with the degree of the region, *i.e.*, the number of involved variables. It is therefore crucial to choose a suitable problem dependent partitioning. Since most of the considered tasks are not decoupled by nature, there are always regions assigned to multiple resources.

Our intuition is illustrated in Fig. 6 for a graphical model with 8 random variables and pairwise regions. The output space is partitioned onto two computers κ_1 and κ_2, and the edges divided onto two machines are highlighted using two parallel lines depicted with blue and red color. In order to guarantee convergence to the optimum of the approximated problem, we are required to enforce that the regions divided onto multiple machines are consistent upon convergence.

More formally let $M(r)$ be the set of machines that region r is assigned to, and let $R(\kappa)$ be the set of regions that are partitioned on machine κ. A region is assigned to a machine if at least one of its variables is assigned to computer κ. Let the potentials and counting numbers be distributed equally across machines, *i.e.*, $\hat{\theta}_r(s_r) = \theta_r(s_r)/|M(r)|$ and $\hat{c}_r = c_r/|M(r)|$. In addition we introduce a junction graph $G_{\mathcal{P}}$ with circular nodes representing the computers and rectangular vertices representing the regions shared between at least two machines. The junction graph for the example in Fig. 6 is illustrated in Fig. 7.

We employ dual decomposition and obtain the following distributed program:

$$\max_{b_r, b_r^\kappa \in \Delta} \sum_{\kappa, r, s_r} b_r^\kappa(s_r)\hat{\theta}_r(s_r) + \sum_{\kappa, r} \epsilon \hat{c}_r H(b_r^\kappa) \tag{11}$$
$$\text{s.t.} \quad \begin{array}{ll} \forall \kappa, r \in R(\kappa), s_r, p \in P(r) & \sum_{s_p \setminus s_r} b_p^\kappa(s_p) = b_r^\kappa(s_r) \\ \forall \kappa, r \in R(\kappa), s_r & b_r^\kappa(s_r) = b_r(s_r). \end{array}$$

Importantly, this program is equivalent to the task given in Eq. (4). The latter constraint is introduced to enforce consistency between the regions assigned to multiple computers. Note that this is the only constraint that couples the individual problems between different computers.

To leverage the structure within the constraint set we change to the dual domain and derive a block-coordinate descent algorithm. We refer the reader to the supplementary material for the detailed derivation and a formal statement regarding the convergence properties. The resulting algorithm is provided in Fig. 8. The procedure to solve the distributed program given in Eq. (11) is divided into two parts: (1) a standard message passing, local on every machine and hence easily parallelizable using graph-coloring, and (2) an exchange of information between different computers enforcing the consistency constraints.

```
Algorithm: Distributed Message Passing Inference
Repeat until convergence
    1) For every $\kappa$ in parallel: iterate over $r$:
        $\forall p \in P(r), s_r \;\; \mu_{p\to r}(s_r) = \epsilon\hat{c}_p \ln \sum_{s_p\backslash s_r} \exp \frac{\hat{\theta}_p(s_p) - \sum_{p'\in P(p)} \lambda_{p\to p'}(s_p) + \sum_{r'\in C(p)\cap\kappa\backslash r} \lambda_{r'\to p}(s_{r'}) + \nu_{\kappa\to p}(s_p)}{\epsilon\hat{c}_p}$   (8)
        $\forall p \in P(r), s_r \;\; \lambda_{r\to p}(s_r) \propto \frac{\hat{c}_p}{\hat{c}_r + \sum_{p\in P(r)} \hat{c}_p} \left(\hat{\theta}_r(s_r) + \sum_{c\in C(r)\cap\kappa} \lambda_{c\to r}(s_c) + \nu_{\kappa\to r}(s_r) + \sum_{p\in P(r)} \mu_{p\to r}(s_r)\right) - \mu_{p\to r}(s_r)$   (9)
    2) Iterate over $r \in G_{\mathcal{P}}$
        $\forall \kappa \in M(r) \;\; \nu_{\kappa\to r}(s_r) = \frac{1}{|M(r)|} \sum_{c\in C(r)} \lambda_{c\to r}(s_c) - \sum_{c\in C(r)\cap\kappa} \lambda_{c\to r}(s_c) + \sum_{p\in P(r)} \lambda_{r\to p}(s_r) - \frac{1}{|M(r)|} \sum_{\kappa\in M(r), p\in P(r)} \lambda_{r\to p}(s_r)$   (10)
```

Fig. 8. A block-coordinate descent algorithm for the distributed inference task.

$$\min_w \sum_{\kappa,(x,s),r} \epsilon\hat{c}_r \ln \sum_{\hat{s}_r} \exp \left(\frac{\hat{\tilde{\phi}}_{(x,s),r}(x,\hat{s}_r) - \sum_{p\in P(r)} \lambda_{(x,s),r\to p}(\hat{s}_r) + \sum_{c\in C(r)} \lambda_{(x,s)c\to r}(\hat{s}_c) + \nu_{(x,s),\kappa\to r}(\hat{s}_r)}{\epsilon\hat{c}_r} \right) - w^\top d + \frac{C}{2}\|w\|_2^2$$

Fig. 9. The distributed and approximated learning task.

This exchange of information corresponds to message passing on the junction graph $G_{\mathcal{P}}$.

The messages ν of the distributed algorithm which are sent between different computers are Lagrange multipliers which arise from the consistency constraint enforcing the beliefs on different machines to be identical upon convergence.

Our distributed algorithm maintains the convergence guarantees irrespective of how frequently we exchange information. The frequency of the exchange only has an impact on the speed of convergence. We expect faster convergence (in terms of number of iterations) if we exchange information at every single iteration. This will however not lead to the best convergence speed in terms of time, due to the communication overhead between computers. Transmission latency will significantly slow down the algorithm. The best tradeoff depends on both, the network latency and the graph structure, as the latter determines the number of messages to be exchanged. Importantly if we choose ϵ, $\hat{c}_r > 0$, the distributed program remains strictly concave. Consequently the dual is smooth and the employed block-coordinate descent methods of both the distributed and the original task converge to the identical solution.

4 Distributed Structured Prediction

Before deriving a distributed learning algorithm we note that it is trivially possible to parallelize the common max-margin or maximum likelihood learning task w.r.t. the number of samples. However, this approach does neither scale to large graphical models requiring large amounts of memory nor is it applicable in a transductive learning setting nor is it effective when using mini-batches containing a small number of samples.

We obtain a distributed learning algorithm by combining the procedure described for deriving the standard learning program with distributed inference. Intuitively, we transform the learning task given in Eq. (6) to the dual domain where we apply the aforementioned decomposition as well as approximations for the entropy and the constraint set. In addition we partition the output space $\mathcal{S}$ of every sample onto different computers κ. This imposes a partitioning of the regions r and hence local beliefs $b^{\kappa}_{(x,s),r}$ if a region is assigned to machine κ. Analogously to the distributed inference approach we are required to enforce consistency between regions upon convergence. This consistency constraint is the only coupling between the different computers, hence we decomposed the dual into almost separable parts. Note that we distribute the counting numbers, the loss and the features equally between the computers, *i.e.*, $\hat{c}_r = c_r/|M(r)|$, $\hat{\ell}_{(x,s),r} = \ell_{(x,s),r}/|M(r)|$ and $\hat{\phi}_{(x,s),k,r} = \phi_{(x,s),k,r}/|M(r)|$.

Following aforementioned ideas we again aim at leveraging the structure given within the constraint set of the decomposed dual. We therefore transform the decomposed approximated dual back to the primal domain. The resulting cost function to be optimized is illustrated in Fig. 9 where we let $\hat{\tilde{\phi}}_{(x,s),r}(x,\hat{s}_r) = \sum_{k:r\in\mathcal{R}_k} w_k \hat{\phi}_{(x,s),k,r}(\hat{s}_r) + \hat{\ell}_{(x,s),r}(\hat{s}_r, s_r)$.

The important differences to standard approximate convex learning originally derived in [6], [7] and illustrated in Fig. 5 are the summation over the computers

```
Algorithm: Distributed Convex Structured Prediction
Repeat until convergence
  1) For every $\kappa, (x_s)$ in parallel: iterate over $r$: $\forall p \in P(r), s_r$
       $\mu_{(x,s),p\to r}(s_r) = \epsilon\hat{c}_p \ln \sum_{s_p\setminus s_r} \exp \frac{\tilde{\hat{\phi}}_{(x,s),p}(s_p) - \sum_{p'\in P(p)} \lambda_{(x,s),p\to p'}(s_p) + \sum_{r'\in C(p)\cap\kappa\setminus r} \lambda_{(x,s),r'\to p}(s_{r'}) + \nu_{(x,s),\kappa\to p}(s_p)}{\epsilon\hat{c}_p}$
       $\lambda_{(x,s),r\to p}(s_r) \propto \frac{\hat{c}_p}{\hat{c}_r + \sum_{p\in P(r)} \hat{c}_p}\left(\tilde{\hat{\phi}}_r(s_r) + \sum_{c\in C(r)\cap\kappa} \lambda_{(x,s),c\to r}(s_c) + \nu_{(x,s),\kappa\to r}(s_r) + \sum_{p\in P(r)} \mu_{(x,s),p\to r}(s_r)\right) - \mu_{(x,s),p\to r}(s_r)$
  2) Iterate over $r \in G_{\mathcal{P}}$: $\forall \kappa \in M(r)$
       $\nu_{(x,s),\kappa\to r}(s_r) = \frac{1}{|M(r)|} \sum_{c\in C(r)} \lambda_{(x,s),c\to r}(s_c) - \sum_{c\in C(r)\cap\kappa} \lambda_{(x,s),c\to r}(s_c) + \sum_{p\in P(r)} \lambda_{(x,s),r\to p}(s_r) - \frac{1}{|M(r)|} \sum_{\kappa\in M(r), p\in P(r)} \lambda_{(x,s),r\to p}(s_r)$
  3) Weight vector update with stepsize $\alpha$ determined to ensure the Armijo rule
       $w \leftarrow w - \alpha \left( \sum_{(x,s),r} \left( \sum_{\hat{s}_r} b_{(x,s),r}(\hat{s}_r)\phi_{(x,s),r}(\hat{s}_r) - \phi_{(x,s),r}(s_r) \right) + Cw \right) \quad (12)$
```

Fig. 10. A block-coordinate descent algorithm for the distributed learning task.

κ and an additional consistency message $\nu_{(x,s),\kappa\to r}(\hat{s}_r)$ to be transferred from machine κ to region r. As for inference, the additional messages ν correspond to Lagrange multipliers for the consistency constraints enforcing distributed region beliefs to agree upon convergence.

In a subsequent step we derive a block-coordinate descent algorithm for the distributed approximated primal problem. Neglecting minor modifications, the updates w.r.t. λ and w are identical to the non-distributed algorithm. To update the Lagrange multipliers ν an additional step is introduced and theoretically only required to be performed occasionally in order to ensure convergence. Given some messages we compute the beliefs

$$b_{(x,s),r}(s_r) \propto \exp\left(\tilde{\hat{\phi}}_{(x,s),r}(x, \hat{s}_r) - \sum_{p\in P(r)} \lambda_{(x,s),r\to p}(\hat{s}_r) + \sum_{c\in C(r)} \lambda_{(x,s)c\to r}(\hat{s}_c) + \nu_{(x,s),\kappa\to r}(\hat{s}_r)\right) / (\epsilon c_r)$$

if $\epsilon c_r > 0$. For $\epsilon c_r = 0$ the beliefs nonzero domain is again given by the maximizing states, analogously to Eq. (5) but employing corresponding messages and potentials. The complete algorithm is summarized in Fig. 10 and we refer the reader to the supplementary material for the derivations and statements regarding convergence guarantees. In short and analogously to inference we obtain an identical solution if the learning task is strictly convex.

Convex approximations of the learning task benefit from interleaving the gradient computation and the line-search while maintaining the convergence guarantees. This is also true when dividing the model onto separate resources and it is required to only transfer information between computers occasionally. Note also that the amount of data to be transmitted between machines depends on the graphical model, *i.e.*, particularly the size of the linking factors, and the number of learned parameters. User attention is required to obtain an efficient procedure. We will show empirically that a model-parallel scenario is on par with sample-parallel training even in the small-scale setting.

5 Implementation Details

Together with this submission we release a software package called *distributed Structured Prediction* (dSP). Note that all the experiments within this submission were conducted with the published framework. We briefly describe the currently implemented features in the following.

Support for higher order regions: Arbitrary Hasse diagrams are taken as input for both learning and inference. Therefore dSP supports junction tree optimization just like ordinary message passing on a factor graph. In addition and at the expense of exponential complexity, models can be manually tightened up to the tree-width.

Support for arbitrary counting numbers: The user is free to specify the counting numbers. Therefore, non-convex Bethe approximations are supported just like convex belief propagation. Parameters determine whether updates for the Lagrange multipliers are interleaved with gradient steps for the model parameters.

Suitable for conditional random fields or structured support vector machines: Changing the parameter ϵ allows optimization of the SSVM or M^{3}N objective (max-margin formulation) for $\epsilon = 0$ just like the CRF cost function (max-likelihood) for $\epsilon = 1$.

Support for high performance computing: The package contains modules that parallelize learning w.r.t. samples on a single machine or w.r.t. the graph coloring of the Hasse diagram. In addition, learning for distributed memory environments supports parallelization by dividing the model or by dividing the samples. Similarly, parameters determine whether inference processes multiple samples concurrently or whether the model is partitioned onto multiple computers. To suit large clusters we utilize standard message passing libraries like OpenMPI or MPICH. To save time for the transmission, we also merge messages of different regions into a single package.

Different input formats: We designed both a textual as well as a binary input file format. While the first is suitable for debugging purposes, the latter targets large scale data sets. Besides file formats a Matlab mex function provides direct access to the modules.

Latent variable models: Although not content of this submission we provide a module to learn from weakly labeled data, *i.e.*, we support latent SSVMs [22] just like hidden CRFs [23] by following [24], [25]. Parallelization w.r.t. samples and by graph coloring is also supported for latent variable models.

6 Experiments

We employed the hybrid approach which uses graph coloring for the learning and inference sub-tasks that are sent by our distributed algorithm to each machine. This allow us to exploit both the advantages of distributed and shared memory architectures within a single algorithm. Counting numbers equal one throughout all experiments.

6.1 Inference

We illustrate the advantages of our approach in the problem of stereo estimation using 16 8-core computers. Towards this goal, we employ the Tsukuba image consisting of 288×369 pixels from the Middlebury data set [26]. The respective Hasse diagram is composed of approximately $315,000$ regions having a label space size of 16 or 256 for unary and pairwise regions respectively. Subsequently we investigate the time required for the iterations before providing results regarding the frequency of transmission of messages between computers. Afterwards we also illustrate the primal-dual gap and visually investigate the results of distributing a task without exchanging messages between computers. We conclude the experiments for inference with a large scale example.

Time per iteration: We investigate the time required for a certain number of iterations. Fig. 11(a) – (c) shows timings for different values of ϵ. Note that the solid magenta line illustrates the baseline algorithm that parallelizes the inference task only onto the available eight cores of a single computer, while the solid lines illustrate the time for a certain number of iterations for the distributed algorithm run on 16 machines and exchanging information only every 1, 2, 5, 10, 25 or 50 iterations. Note that the distributed algorithm is significantly faster than the single machine shared memory counterpart. The speedups are given in Tab. 1. Due to computational benefits of finding the maximum within a vector rather than computing the soft-max we also note that inference with $\epsilon = 0$ is slightly faster.

	50	25	10	5	2	1
0	16.60	15.74	14.95	13.51	10.48	7.67
0.01	15.57	15.27	14.34	13.19	10.87	8.32
0.10	15.38	15.10	14.23	13.24	10.76	8.24

TABLE 1
Inference speedup after 200 iterations for different ϵ (rows) and information exchange rates (columns).

Frequency of transmission: We next again use the distributed approach which divides the graph into 16 equally sized partitions, and investigate how often to transmit information between the different computers. Fig. 12(a) – (c) illustrates the dual energy w.r.t. time. Depending on ϵ and for the investigated data it is beneficial to transfer messages between different computers only every two or only every five iterations. We note that this obviously depends largely on the problem at hand and on the connectivity between the computers being in our case a 4-connected grid graph and a standard local area network connection.

Primal-dual gap: The primal and dual scores for different $\epsilon \in \{0, 0.01, 0.1\}$ are illustrated in Fig. 13(a) – (c). Note that our distributed block-coordinate descent algorithm guarantees the monotonicity of the dual energy. This is generally not the case for the primal energy. We emphasize that due to only 200 iterations the primal-dual gap is not yet necessarily 0, even for $\epsilon > 0$.

Simple distributed baseline: We refer the reader to Fig. 14 for visual results comparing the exchange of information with an approach that distributes the problem without exchanging information. For this experiment we divided a disparity computation task for the image of a tree onto nine computers. We observe clear artifacts when not exchanging messages between the sub-problems.

Large scale graphs: To illustrate the large scale behavior we perform disparity map estimation with 283 labels on the 12 MPixel image illustrated in Fig. 1. The Hasse diagram consists of more than $36,000,000$ regions with 12 million regions having about 250 disparity levels while 24 million regions have more than $80,000$ states. Note that the considered pairwise Markov random field has a complexity quadratic in the disparity labels.

Scaling: In a next experiment we evaluate the scaling of the general inference implementation with successively larger input. We start with disparity map

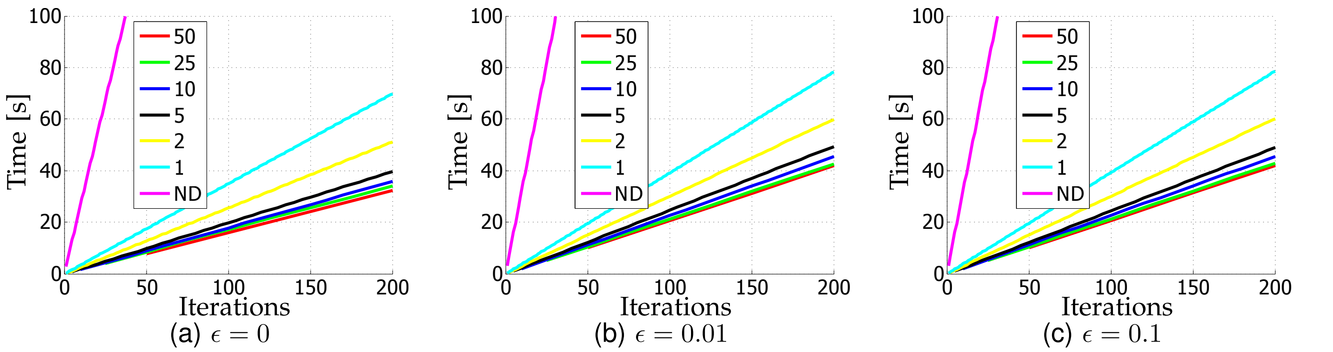

Fig. 11. Comparing the time of non-distributed ('ND') inference (magenta) with the distributed inference algorithm exchanging information at different frequencies when considering a given number of iterations.

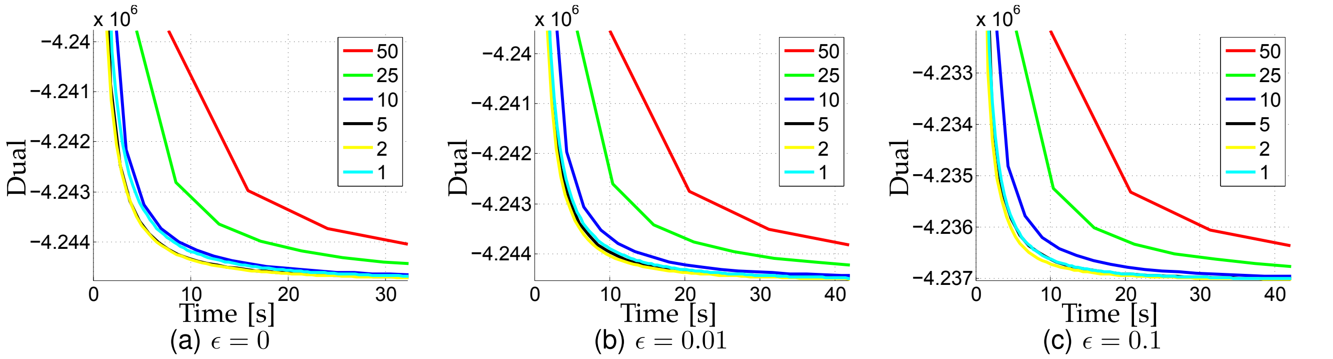

Fig. 12. Dual of the distributed inference algorithm over time with information exchange at different frequencies.

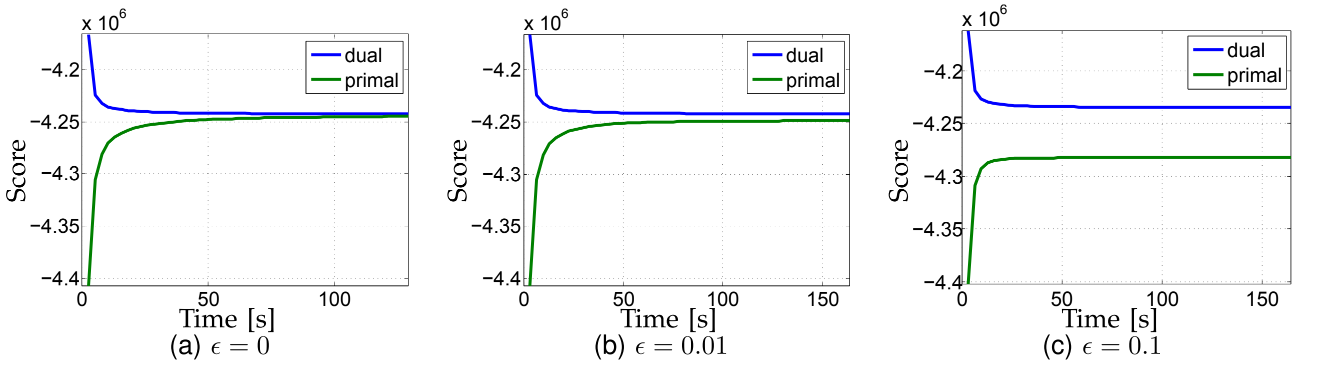

Fig. 13. Primal and dual scores of the inference algorithm for different ϵ.

estimation for an image of 26k pixels each having 10 states and increase it to more than 2 megapixels each having 116 states. We compare the distributed implementation ('Dist') tiling the images into 9 roughly equally sized partitions while running on 9 equally powerful machines to a non-distributed approach running on a single computer ('Sing'). The results are visualized in a double-logarithmic plot in Fig. 15. We observe that our approach obviously does not change the algorithmic complexity. However the distributed method is increasingly faster for larger problems. From about twice faster for 26k sized images to almost 7 times faster for 0.5 megapixels. The observations are consistent across our tests for $\epsilon \in \{0, 0.01, 0.1, 1\}$.

6.2 Learning

We are interested in answering similar questions when considering the distributed learning task. To this end we consider a denoising problem. We are given a 4 bit ground truth image, *i.e.*, 16 gray scale levels, and a set of noisy observations as illustrated for one example in Fig. 16. Our two dimensional feature vector measures on the one hand the difference of the noisy observation to the 16 gray scale levels. On the other hand we encode smoothness by employing a truncated linear pairwise potential. We aim at learning a linear combination of those two features.

We want to emphasize again that there are two main paradigms for distributed learning, data parallelism and model parallelism. Both are supported by the provided library since the preferable method obviously depends on the particularities of the data and the corresponding graphical model. *E.g.*, if we work in a transductive setting or if the number of samples (in a mini-batch) is small compared to the model size, we typically prefer model parallelism. Note that this

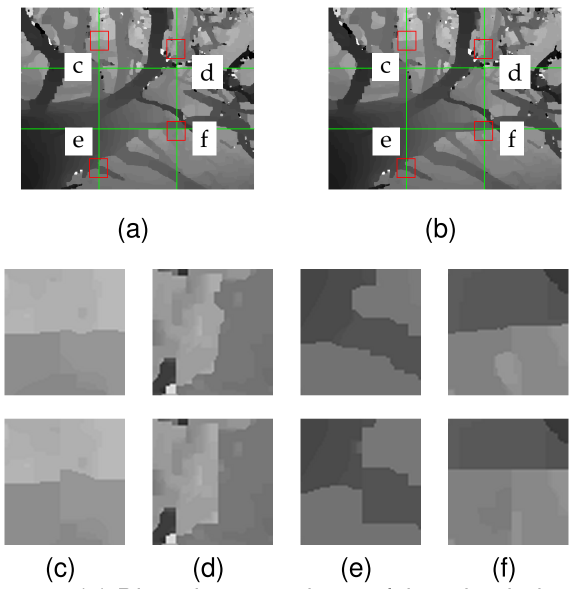

Fig. 14. (a) Disparity map when solving nine independent sub-problems. Frame boundaries are depicted. (b) Disparity map when exchanging messages every 10 iterations. (c)-(f) zoomed view of the highlighted parts in (a) (bottom) and (b) (top).

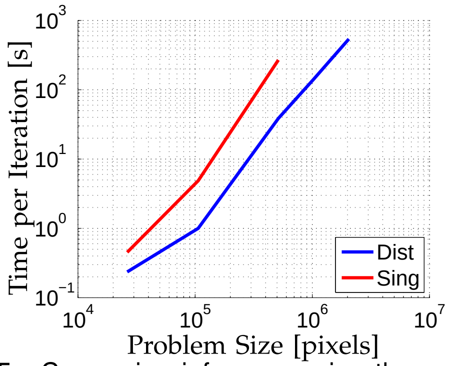

Fig. 15. Comparing inference using the general distributed approach ('Dist') to a single computer implementation ('Sing') for $\epsilon = 1$.

is for example the case when combining deep learning techniques with structured prediction [27], [28]. In contrast, if the graphical models are small but the number of samples is abundant, we recommend usage of data parallelism. To demonstrate this effect we first investigate the tradeoff between data parallelism and model parallelism more carefully before we take a closer look at the effects of the message exchange frequency on the primal value as well as the speed-ups in a second experiment.

Data parallelism vs. model parallelism: This section is dedicated to answer under which circumstances one form of parallelism outperforms the other. To clarify the tradeoffs we compare a data parallel setting where the samples are distributed onto 9 machines each having 8 cores with a model parallel setting where each grid-graph is partitioned into 9 sub-graphs, each being assigned to one of our 9 machines. In the model parallel setting the samples are processed sequentially while message passing is parallelized using a graph-coloring technique. In Fig. 17(a) we illustrate the time ratio of 50 training iterations of model parallelism over sample parallelism for a temperature parameter $\epsilon = 0$. The number of training instances is specified on the x-axis. A number smaller than 1 indicates preference for model parallelism. The solid lines correspond to graphical models of size 128×128 while the dashed lines illustrate the results for model sizes of 64×64. The colors indicate the number of message passing iterations before exchanging messages between the 9 computers. As expected we observe model parallelism to be beneficial for less frequent message exchanges, larger sized samples and fewer training instances. The latter is particularly useful when considering stochastic gradient descent training with mini-batches typically containing around or less than 100 samples. Note also that model parallelism is not only beneficial if we cannot fit a single example into memory, *e.g.*, we are better off using model parallelism when processing 100 samples of size 128×128, independent of the message exchange frequency.

Fig. 16. A 4 bit ground truth image and one of the noisy observation given for training.

	1	2	5	10	25
0.00	0.58	0.51	0.47	0.45	0.45
0.10	0.62	0.55	0.50	0.48	0.48
1.00	0.68	0.64	0.57	0.55	0.54

TABLE 2
Time ratio of model parallelism over sample parallelism for different exchange frequencies and temperature parameter ϵ using 100 training samples of size 128×128.

In a next experiment we investigate the influence of the temperature parameter ϵ on the tradeoff between data parallelism and model parallelism. The results are provided in Fig. 17. We observe the trends to be very robust. This is also apparent when quantitatively investigating the ratios for 100 training samples of size 128×128 given in Tab. 2.

Frequency of transmission: In a next experiment we investigate the occurring effects when exchanging information between machines at different frequencies for different values of ϵ. We use a training set of 100 samples each of size 128×128 for this experi-

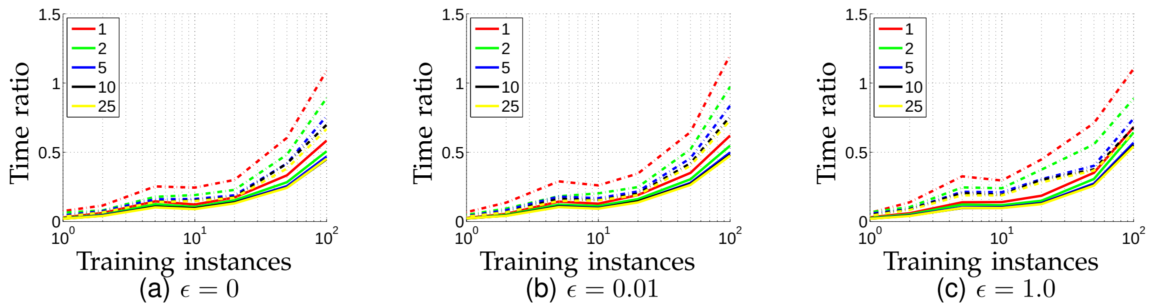

Fig. 17. Time ratio of model parallelism over sample parallelism for a different number of training instances having size 64×64 (dashed) or 128×128 (solid), exchange frequencies and temperature parameter ϵ.

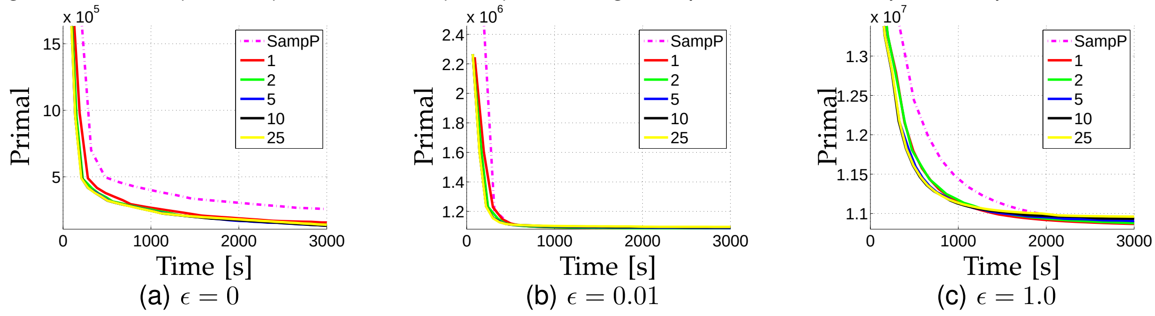

Fig. 18. Primal over time for different exchange frequencies and temperature parameter ϵ.

ment. The results are provided in Fig. 18. Depending on the choice of the temperature parameter ϵ we observe different message exchange frequencies to be beneficial. We also compare the primal value obtained using model parallel optimization to the one achieved with sample parallelism which we refer to using the abbreviation 'SampP.' Note that the primal value corresponds to the annealed log-likelihood, hence illustrating the model fit to training data. We observe model parallelism to be very beneficial for the temperature parameter $\epsilon = 0$.

To conclude the experimental section we showed the benefits and disadvantages when distributing a given problem onto multiple machines. An additional notable advantage for model parallelism are the distributed resource requirements. Since only part of the task is stored on a machine, memory requirements per computer are lower.

7 Discussion

For over a decade now, much effort has been devoted to finding efficient, yet (provably) convergent inference algorithms for graphical models. Graph-cuts and message-passing algorithms are amongst the most popular inference techniques that have been applied to solve computer vision problems. Submodular functions have been constructed to target these problems, as graph-cuts are exact in this setting. We refer the reader to [2] for a detailed description on optimality conditions. To solve multi-label problems, techniques that rely on selecting a set of moves that solve at each step a binary problem have been proposed, *e.g.*, α-expansion [29] or fusion moves [30]. However, while the individual moves can be solved to optimality, the algorithm is not guaranteed to find the global optimum.

Several approaches have also been developed to parallelize and distribute graph-cut algorithms. Strandmark and Kahl [31] and Shekhovtsov and Hlavac [32] proposed a parallel and distributed graph-cut approach using dual decomposition. Their method facilitates computation of larger problems by partitioning the model onto multiple machines. Similar to their intention we aim at assigning the inference task to multiple computers. Contrasting their work, we presented a decomposition method for message-passing algorithms to ensure applicability for non-submodular potentials. This is important since non-submodular potentials arise in applications such as image editing [33] and segmentation [34].

Message passing algorithms like belief propagation (BP) are exact when the underlying graphical model is a tree without regard of the type of potentials [17]. To allow for faster computation, Felzenszwalb *et al.* [35] proposed a red-black graph coloring algorithm for parallelizing BP on grid structures. Here, we extend this strategy to general graphs and Hasse diagrams using a greedy graph coloring algorithm, and employ this extension for local computations within each machine. The main drawback of the BP algorithm is that it is not guaranteed to converge in many cases of interest.

Convexity was extensively utilized in recent years to develop message passing algorithms that are guaranteed to converge [36], [37], [38], [39], [40], [41], [16], [15], [20]. However, as any other message-passing algorithm, the main limitations are memory requirements and computational complexity. Although all these algorithms can be trivially used in a parallel or distributed manner (also referred as "inherently parallel") they lose their convergence guarantees. Therefore, [14] introduced a parallel convex sum-product algorithm, which provably gives the optimal solution for strictly concave entropies. This algorithm differs from ours in important aspects: it propagates a $(1/n)$-fraction of information through its messages, where n is the number of nodes in the graph. It therefore converges slowly even for small graphs, *e.g.*, if $n = 100$, and it is numerically unstable for large graphs which are the focus of this work. In contrast, we use consistency messages to propagate a constant fraction of the information. Moreover, this work is a natural extension of the distributed inference algorithm in [8] to the more general region graphs visualized by Hasse diagrams.

A few years ago, Low *et al.* [42] presented GraphLab, a framework for efficient and provably convergent parallel algorithms. They show impressive results on typical machine learning tasks such as belief propagation by improving on the MapReduce abstraction. Unfortunately, their original implementation assumed that all the data is stored in shared-memory, which made it infeasible to apply an early version of GraphLab to large scale problems. For example if we were to use this early version for the largest example shown in this paper we will need a computer with 50 gigabyte of memory. The shared memory assumption is severe as it does not allow efficient distribution of the task at hand to multiple machines. In contrast, we distribute the memory requirements such that this is no longer a burden.

More recently Low *et al.* [43] and Gonzalez *et al.* [44] also presented algorithms for distributed memory environments which are now part of Dato. Their methodology does not take into account the differences between variables on the same computer and variables that can only be reached via a slow network. In contrast we employ dual decomposition to minimize the required transmissions between different computers while maintaining convergence guarantees.

Learning structured predictors has played a significant role in computer vision. CRFs were first described in [5] and SSVMs, M^{3}Ns were developed in [45], [3], [4]. These algorithms are "inherently parallel" as they can all partition the different training instances to different computing nodes. In contrast, our work also provides the means to distribute the inference process of individual training instances.

In short the presented algorithms also extend [8], [9] in that we now support the more general Hasse diagrams. We further extend [6] in distributing the problem to minimize transmission overhead when considering distributed memory environments.

8 Conclusion

We have derived a *distributed message passing* algorithm that is able to do inference in *large scale graphical models* by dividing the computation and memory requirements onto multiple machines. Importantly, the convergence and optimality guarantees and properties of convex belief propagation are preserved by introducing new types of messages that are sent between the different machines. We have demonstrated the effectiveness of our approach in the task of stereo reconstruction from high-resolution imagery as well as denoising. The main benefit of our approach arises from the use of multiple computers. Thus we expect that running our algorithm within even larger clusters, such as *Amazon EC2*, will result in orders of magnitude further speed up.

While this work assumes the user to provide a partitioning of the problem, we leave the task of finding the best partitioning to future research.

To reproduce the experiments within this submission on your own set of computers we released *dSP* sources and all the data on http://alexander-schwing.de.

References

[1] S. Brooks, A. Gelman, G. L. Jones, and X. Meng, *Handbook of Markov Chain Monte Carlo*. Chapman & Hall, 2011.
[2] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts?" *PAMI*, 2004.
[3] B. Taskar, C. Guestrin, and D. Koller, "Max-Margin Markov Networks," in *Proc. NIPS*, 2003.
[4] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large Margin Methods for Structured and Interdependent Output Variables," *JMLR*, 2005.
[5] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. ICML*, 2001.
[6] T. Hazan and R. Urtasun, "A Primal-Dual Message-Passing Algorithm for Approximated Large Scale Structured Prediction," in *Proc. NIPS*, 2010.
[7] O. Meshi, D. Sontag, T. Jaakkola, and A. Globerson, "Learning Efficiently with Approximate inference via Dual Losses," in *Proc. ICML*, 2010.
[8] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun, "Distributed Message Passing for Large Scale Graphical Models," in *Proc. CVPR*, 2011.
[9] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun, "Distributed Structured Prediction for Big Data," in *NIPS Workshop on Big Learning*, 2012.
[10] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun, "Globally Convergent Dual MAP LP Relaxation Solvers using Fenchel-Young Margins," in *Proc. NIPS*, 2012.
[11] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun, "Globally Convergent Parallel MAP LP Relaxation Solver using the Frank-Wolfe Algorithm," in *Proc. ICML*, 2014.
[12] W. Wiegerinck and T. Heskes, "Fractional belief propagation," in *Proc. NIPS*, 2003.
[13] Y. Weiss, C. Yanover, and T. Meltzer, "MAP Estimation, Linear Programming and Belief Propagation with Convex Free Energies," in *Proc. UAI*, 2007.
[14] T. Hazan and A. Shashua, "Convergent message-passing algorithms for inference over general graphs with convex free energy," in *Proc. UAI*, 2008.
[15] T. Meltzer, A. Globerson, and Y. Weiss, "Convergent message passing algorithms - a unifying view," in *Proc. UAI*, 2009.
[16] T. Hazan and A. Shashua, "Norm-Product Belief Propagation: Primal-Dual Message-Passing for LP-Relaxation and Approximate-Inference," *Trans. on Information Theory*, 2010.

[17] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.
[18] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *Trans. on Information Theory*, 2005.
[19] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[20] T. Hazan, J. Peng, and A. Shashua, "Tightening Fractional Covering Upper Bounds on the Partition Function for High-Order Region Graphs," in *Proc. UAI*, 2012.
[21] D. P. Bertsekas, *Nonlinear Programming - Second Edition*. Athena Scientific, 1999.
[22] C.-N. Yu and T. Joachims, "Learning Structural SVMs with Latent Variables," in *Proc. ICML*, 2009.
[23] A. Quattoni, S. Wang, L.-P. Morency, M. Collins, and T. Darrell, "Hidden-state Conditional Random Fields," *PAMI*, 2007.
[24] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun, "Efficient Structured Prediction with Latent Variables for General Graphical Models," in *Proc. ICML*, 2012.
[25] A. G. Schwing, "Inference and learning algorithms with applications to 3D indoor scene understanding," Ph.D. dissertation, ETH Zurich, 2014.
[26] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *IJCV*, 2002.
[27] L.-C. Chen*, A. G. Schwing*, A. L. Yuille, and R. Urtasun, "Learning Deep Structured Models," in *Proc. ICML*, 2015, * equal contribution.
[28] A. G. Schwing and R. Urtasun, "Fully Connected Deep Structured Networks," 2015, available on http://arxiv.org/abs/1503.02351.
[29] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *PAMI*, 2001.
[30] V. Lempitsky, C. Rother, and A. Blake, "Logcut-efficient graph cut optimization for Markov random fields," in *Proc. ICCV*, 2007.
[31] P. Strandmark and F. Kahl, "Parallel and distributed graph cuts by dual decomposition," in *Proc. CVPR*, 2010.
[32] A. Shekhovtsov and V. Hlavac, "A Distributed Mincut/Maxflow Algorithm Combining Path Augmentation and Push-Relabel," in *Proc. EMMCVPR*, 2011.
[33] Y. Pritch, E. Kav-Venaki, and S. Peleg, "Shift-map image editing," in *Proc. ICCV*, 2010.
[34] P. Kohli, L. Ladický, and P. H. S. Torr, "Robust higher order potentials for enforcing label consistency," *IJCV*, 2009.
[35] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *IJCV*, 2006.
[36] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *PAMI*, 2006.
[37] T. Heskes, "Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies," *J. of AI Research*, 2006.
[38] A. Globerson and T. S. Jaakkola, "Fixing max-product: convergent message passing algorithms for MAP relaxations," in *Proc. NIPS*, 2007.
[39] N. Komodakis and N. Paragios, "Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles," in *Proc. ECCV*, 2008.
[40] N. Komodakis, N. Paragios, and G. Tziritas, "MRF Energy Minimization & Beyond via Dual Decomposition," *PAMI*, 2010.
[41] T. Werner, "A linear programming approach to max-sum problem: A review," *PAMI*, 2007.
[42] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new parallel framework for machine learning," in *Proc. UAI*, 2010.
[43] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud," *PVLDB*, 2012.
[44] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," in *Proc. OSDI*, 2012.
[45] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *Proc. ACL*, 2002.

Alexander G. Schwing is a postdoctoral fellow in the Computer Science department at University of Toronto and also supported by the Fields Institute. In 2014 he completed his PhD at ETH Zurich which awarded his thesis with the ETH medal. His area of research is inference and learning algorithms for structured distributions.

Tamir Hazan received his PhD from the Hebrew University of Jerusalem (2010) and he is currently an assistant professor at the University of Haifa, Israel. Tamir Hazans research describes efficient methods for reasoning about structured models. His work on random perturbations was presented as a best paper shortlist at ICML 2012 and in the machine learning best papers track at AAAI 2012. Tamir Hazans research also includes the primal-dual norm-product belief propagation algorithm which received a best paper award at UAI 2008.

Marc Pollefeys is a full professor in the Dept. of Computer Science of ETH Zurich since 2007. Before that he was on the faculty at the University of North Carolina at Chapel Hill. He obtained his PhD from the KU Leuven in Belgium in 1999. His main area of research is computer vision, but he is also active in robotics, machine learning and computer graphics. Dr. Pollefeys has received several prizes for his research, including a Marr prize, an NSF CAREER award, a Packard Fellowship and a European Research Council Grant. He is the author or co-author of more than 250 peer-reviewed publications. He was the general chair of ECCV 2014 in Zurich and program chair of CVPR 2009. He is a fellow of the IEEE.

Raquel Urtasun is an assistant professor at the Dept. of Computer Science, University of Toronto. From 2009-2014 she was an assistant professor at TTI-Chicago, a philanthropically endowed academic institute located in the campus of the University of Chicago. She was a visiting professor at ETH Zurich during the spring semester of 2010. Previously, she was a postdoctoral research scientist at UC Berkeley and ICSI and a postdoctoral associate at the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. Raquel Urtasun completed her PhD at the Computer Vision Laboratory, at EPFL, Switzerland in 2006 working with Pascal Fua and David Fleet at the University of Toronto. She has been area chair of multiple machine learning and vision conferences (i.e., NIPS, UAI, ICML, CVPR, ECCV, ICCV), she is on the editorial board of the International Journal of Computer Vision (IJCV), and served in the committee of numerous international conferences. She has won multiple awards including the best paper runner up at CVPR 2013, the Connaught New Researcher Award in 2014 and the Google Faculty Research Award and the Ministry of Research & Innovation Early Research Award in 2015. Her major interests are statistical machine learning, computer vision and robotics, with a particular interest in structured prediction and its application to autonomous driving.