Distributed Message Passing for Large Scale Graphical Models

Alexander Schwing
ETH Zurich

Tamir Hazan
TTI Chicago

Marc Pollefeys
ETH Zurich

Raquel Urtasun
TTI Chicago

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

Abstract

In this paper we propose a distributed message-passing algorithm for inference in large scale graphical models. Our method can handle large problems efficiently by distributing and parallelizing the computation and the memory requirements. The convergence and optimality guarantees of recently developed message-passing algorithms are preserved by introducing new types of consistency messages, sent between the distributed computers. We demonstrate the effectiveness of our approach in the task of stereo reconstruction from high-resolution imagery, and show that inference is possible with more than 200 labels in images larger than 10 MPixel.

1. Introduction

A wide variety of vision problems can be naturally formulated as discrete labeling problems in an *undirected graphical model*. Among the most popular examples are stereo and segmentation. Markov random fields (MRFs) provide a principled framework for modeling and solving these problems. Two main techniques have been developed to perform *inference* in these graphical models, namely, message-passing algorithms, e.g., belief propagation [19] and graph cuts [1].

These two techniques are complementary; one would like to use one or the other depending on the potentials and structure of the graph. For example, graph cuts are optimal for sub-modular functions, while message-passing is optimal when the graph structure is a tree. Here we focus on message passing algorithms which have proven very effective to solve vision tasks such as super-resolution [3], stereo reconstruction [25], segmentation [14], denoising [12], and inpainting [15]. However, the main underlying limitations for their application to real-world problems are memory and computation. This is important since nowadays high-resolution cameras are easily available, and the resulting problems are too large to be handled by traditional message-passing algorithms.

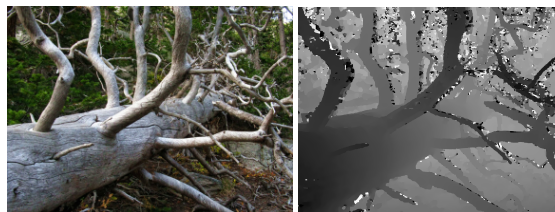We are interested in making message-passing algorithms practical for large scale graphical models. We focus on the paradigm of distributed systems, such as *Amazon EC2*, and develop a new algorithm that is able to distribute and parallelize the computation and memory requirements while conserving the convergence and optimality guarantees of recently developed message-passing algorithms [7, 28, 4, 10, 6].

Our approach is based on the simple principle that computation can be done in parallel by partitioning the graph and imposing agreement between the beliefs in the boundaries. Thus, we split the graph-based optimization program into several local optimization problems (one per machine) that are solved in parallel. This scheme forces introduction of additional Lagrange multipliers, which are sent as messages between machines. This results in a new distributed message-passing algorithm that preserves the convergence guarantees of existing methods. Since standard computers have a multicore CPU, we also parallelize each local optimization problem by using a greedy vertex coloring. In particular, we apply an extension of the red-black scheme of Felzenszwalb and Huttenlocher [2] to general graphs by exploiting graph coloring and performing the computation of nodes having the same color in parallel. Thus our approach benefits from two levels of parallelization, across machines and within the cores of each machine, while conserving the theoretical guarantees. Code will be provided upon acceptance.

We demonstrate the effectiveness of our approach in the task of stereo reconstruction from high-resolution imagery and show that inference is possible in large images (> 10 MPixel) with large number of labels (> 200). See Fig. 1 for an illustration. In the following, we first review related work, we then present our new distributed message-passing algorithm, show experimental evaluation and conclude with future directions.

2. Related Work

For over a decade now, much effort has been devoted to finding efficient, yet (provably) convergent inference algorithms for graphical models. The literature contains many impressive heuristic approximate inference algorithms that obtain reasonable solutions fast, but do have neither convergence nor optimality guarantees [2, 20]. Our focus is different, we target provable convergence while still being computationally tractable. In particular, we present an algorithm that parallelizes convex belief propagation and conserves its convergence and optimality guarantees.

Graph-cuts and message-passing algorithms are amongst the most popular inference techniques that have been applied to solve computer vision problems. Sub-modular functions have been constructed to target these problems, as graph-cuts are exact in this setting. We refer the reader to [9] 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 developed, *e.g.*, α-expansion [1], fusion moves [13]. However, while the individual moves can be solved to optimality, the algorithm is not guaranteed to find the global optimum.

Belief propagation (BP) is exact when the underlying graphical model is a tree without regard of the type of potentials [19]. Except for some special cases [18], neither convergence nor optimality guarantees are available for graphs with cycles. Recently, a new message passing algorithm, named convex belief propagation (convex BP) [6], was shown to always converge. It gives the optimal solution for strictly concave entropies, or the best assignment if the optimal beliefs are without ties. However and just like any other message-passing algorithm, the main limitations for applying it to real-world problems are memory and computational complexity.

To allow for faster computation, Felzenszwalb and Huttenlocher [2] proposed a red-black graph coloring algorithm for parallelizing BP on grid structures. In this paper we apply an extension of this strategy to general graphs using a greedy graph coloring algorithm. We employ this extension in the local computations within each machine.

Several approaches have also been developed to parallelize graph cuts algorithms. Strandmark and Kahl [24] proposed a parallel and distributed graph cut approach using dual decomposition. Their method facilitates computation of larger problems by splitting the model across multiple machines. Similar to their intention we aim at assigning the inference task to multiple computers. Contrasting their work, we will derive 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 [20] and segmentation [8].

Low *et al*. [16] presented GraphLab, a framework for efficient and provably convergent parallel algorithms. They show impressive results on typical machine learning tasks like BP by improving on the MapReduce abstraction. Unfortunately, their implementation assumes that all the data is stored in shared-memory, which makes it infeasible to apply GraphLab to large scale problems. For example if we were to use GraphLab for the largest example shown in this paper we will need a computer with 50 gigabyte (GB) 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.

Thus we combine the findings of the aforementioned papers and

- Split the message passing task at hand into several local optimization problems that are solved in parallel.
- To ensure convergence we force the local tasks to communicate occasionally.
- At the local level we parallelize the message passing algorithm using a greedy vertex coloring.

3. A review on message passing algorithms

Inference in graphical models considers distributions $p(x_1, ..., x_n)$ over a set of random variables that factor into a product of potential functions, each defined over a small number of variables, *i.e.*, *factors*. More formally, let $x_1, ..., x_n$ be the realizations of n discrete random variables, with $x_i \in \{1, ..., n_i\}$. The joint distribution factors into a product of non-negative functions (potentials)

$$p(x_1, ..., x_n) = \frac{1}{Z} \prod_{i=1}^{n} \psi_i(x_i) \prod_{\alpha=1}^{m} \psi_\alpha(\mathbf{x}_\alpha), \qquad (1)$$

where the functions $\psi_i(x_i)$ represent "local evidence" or prior data on the states of x_i. The m functions $\psi_\alpha(\mathbf{x}_\alpha)$ have arguments $\mathbf{x}_\alpha$ that are some subset of $\{x_1, ..., x_n\}$ and Z is a normalization constant, typically referred to as the *partition function*. The factorization structure above defines a *hypergraph* whose nodes represent the n random variables and the subsets of variables $\mathbf{x}_\alpha$ correspond to its hyperedges.

A convenient way to represent hypergraphs is by a bipartite graph with one set of nodes corresponding to the original nodes of the hypergraph and the other set consisting of its hyperedges. In the context of graphical models such a bipartite graph representation is referred to as a *factor graph* [11] with *variable nodes* representing $\psi_i(x_i)$ and a *factor node* for each function $\psi_\alpha(\mathbf{x}_\alpha)$. An edge connects a variable node i with factor node α if and only if $x_i \in \mathbf{x}_\alpha$, *i.e.*, x_i is an argument of ψ_α. We adopt the terminology where $N(i)$ stands for all factor nodes that are neighbors

of variable node i, *i.e.*, all the nodes α for which $x_i \in \mathbf{x}_\alpha$. $N(\alpha)$ stands for all variable nodes that are neighbors of factor node α.

In this paper we focus on computing the maximum a-posteriori (MAP) assignment, which is the task of finding a state for each x_i that brings the maximal value to the joint probability $p(x_1, ..., x_n)$, *i.e.*

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} \prod_{i=1}^n \psi_i(x_i) \prod_{\alpha=1}^m \psi_\alpha(\mathbf{x}_\alpha). \qquad (2)$$

In the following we first reformulate the MAP problem as integer linear program. Taking the logarithm, namely $\theta_\alpha = \ln \psi_\alpha$ and $\theta_i = \ln \psi_i$, we rephrase the MAP problem in (2) by its linear form. Also, we introduce indicator variables $b_i(x_i)$ for each individual variable, and additional indicator variables $b_\alpha(\mathbf{x}_\alpha)$ for the factors. We thus define the integer linear program

$$\max \sum_{\alpha, \mathbf{x}_\alpha} b_\alpha(\mathbf{x}_\alpha)\theta_\alpha(\mathbf{x}_\alpha) + \sum_{i,x_i} b_i(x_i)\theta_i(x_i) \qquad (3)$$

subject to:

$$b_i(x_i), b_\alpha(\mathbf{x}_\alpha) \in \{0,1\}, \ \sum_{\mathbf{x}_\alpha} b_\alpha(\mathbf{x}_\alpha) = 1, \sum_{x_i} b_i(x_i) = 1$$

$$\forall i, x_i, \alpha \in N(i), \sum_{\mathbf{x}_\alpha \backslash x_i} b_\alpha(\mathbf{x}_\alpha) = b_i(x_i)$$

This integer linear program is equivalent to the MAP problem in (2), and is hence NP-hard [23]. We obtain a linear programming relaxation by replacing the integer constraints with non-negativity constraints. The relaxed problem maximizes a linear objective over a set of linear constraints. If the solution to the LP happens to be integer, it corresponds to the MAP estimate.

Typically, primal and dual LP solvers for the relaxation in (3) need to explicitly consider the boundary of the feasible set. The probability simplex constraints over $b_i(x_i), b_\alpha(\mathbf{x}_\alpha)$ involve inequalities, which usually increase the computational complexity of the solver. To avoid this computational overhead we use the fact that inference commonly considers the probability simplex constraints, and uses the entropy barrier function over these constraints, *i.e.* $H(\mathbf{b}_\alpha), H(\mathbf{b}_i)$, defined by

$$H(\mathbf{p}) = \begin{cases} -\sum_x p(x) \ln p(x) & p(x) \geq 0, \sum_x p(x) = 1 \\ -\infty & \text{otherwise} \end{cases}$$
$$(4)$$

The entropy barriers are appealing since they are Legendre-type functions. In particular, they are strictly concave and therefore their duals are smooth (cf. [21], chapter 26). Furthermore, they are bounded for every probability distribution, thus making their dual function finite. Compared to other barrier functions, such as the log-barrier, the

derivatives of the entropy function are linear in the log-space, thus correspond to closed-form updates in a dual block descent algorithm. Consequently, we define the LP relaxation with the entropy barrier[1]:

$$\max \sum_{\alpha, \mathbf{x}_\alpha} b_\alpha(\mathbf{x}_\alpha)\theta_\alpha(\mathbf{x}_\alpha) + \sum_{i,x_i} b_i(x_i)\theta_i(x_i) \qquad (5)$$

$$+\epsilon \left(\sum_\alpha c_\alpha H(\mathbf{b}_\alpha) + \sum_i c_i H(\mathbf{b}_i) \right)$$

subject to:

$$\forall i, x_i, \alpha \in N(i), \sum_{\mathbf{x}_\alpha \backslash x_i} b_\alpha(\mathbf{x}_\alpha) = b_i(x_i)$$

For $\epsilon = 1$, the program in (5) describes the variational approach for approximating the partition function Z and the marginal probabilities $p(\mathbf{x}_\alpha) = \sum_{\mathbf{x} \backslash \mathbf{x}_\alpha} p(\mathbf{x})$ and $p(x_i) = \sum_{\mathbf{x} \backslash x_i} p(\mathbf{x})$ of the probability distribution in (1). In this setting the program in (5) corresponds to the *approximate free energy*, and the entropy is referred to as the *fractional entropy approximation*. In particular, if c_i, c_α correspond to the *tree-reweighted entropy*, *i.e.* c_α is the weighted number of spanning trees that pass through edge α and $c_i = 1 - \sum_{\alpha \in N(i)} c_\alpha$, then the program represents an upper bound to the log-partition function. Whenever the graph has no cycles, and the entropy terms correspond to the *Bethe entropy*, *i.e.* $c_\alpha = 1, c_i = 1 - |N(i)|$, the program in (5) is an exact representation of the log-partition function, or equivalently the free energy. In particular, its optimal arguments are the marginals of the probability in (1), hence it serves as a low-dimensional representation of the Gibbs-Helmholtz free energy:

$$\begin{aligned} -F(\mathbf{p}) &= \sum_{i,x_i} \theta_i(x_i)p(x_i) + \sum_{\alpha, \mathbf{x}_\alpha} \theta_\alpha(\mathbf{x}_\alpha)p(\mathbf{x}_\alpha) + H(\mathbf{p}) \\ &= -E(\mathbf{p}) + H(\mathbf{p}) \end{aligned}$$

The linear term $E(\mathbf{p})$ is often referred to as the energy term.

For non-negative c_i, c_α, the program in (5) is concave, therefore can be solved with any standard solver. However, these solvers do not exploit the structure of the graph, thus suffer from memory constraints when dealing with medium-scale problems [29]. Recently, message-passing algorithms were introduced for solving these problems [27, 26, 28, 4, 7, 10]. Since these algorithms send messages along the edges of the graphical model, they handle memory more efficiently and can deal with larger problems. However, even when using message-passing solvers, the program in (5) becomes infeasible in terms of memory

---

[1]For $\epsilon = 0$ we define the function $\epsilon H(\mathbf{p})$ to be the indicator function over the probability simplex constraints for $\mathbf{p}$. This way the program in (5) equals to the relaxation of program (3) for $\epsilon = 0$
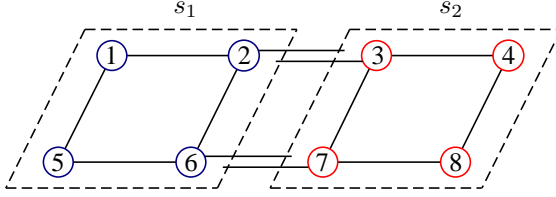
Figure 2. The distributed architecture: $\mathcal{P}$ is the partition of the 8 nodes onto two computers. Its elements are $s_1 = \{1, 2, 5, 6\}$ and $s_2 = \{3, 4, 7, 8\}$. The partition graph $G_\mathcal{P}$ is the multigraph whose vertices are s_1 and s_2 and whose edges are $(2, 3)$ and $(6, 7)$. The edges $(2, 3)$ and $(6, 7)$ are shared between graphs, thus $N_\mathcal{P}(\alpha) = \{s_1, s_2\}$. The vertex s_1 has two edges in $G_\mathcal{P}$, i.e. $N_\mathcal{P}(s_1) = \{(2, 3), (6, 7)\}$.

and computation when dealing with large graphs, *e.g.*, those that arise by the use of high-resolution imagery. In the next section we will introduce a distributed algorithm that has the same guarantees as convex BP.

4. Distributed Convex Belief Propagation

We seek a distributed algorithm for minimizing the inference program in (5), that leverages the graph structure, *i.e.* sends messages along the edges of the graphical model. Moreover, we want to partition the vertices of the graphical model to disjoint subgraphs, such that each computer solves independently a variational program with respect to its subgraph. The distributed solutions are then integrated through message-passing between the subgraphs, thus preserving the consistency of the graphical model. This distributed algorithm extends convex BP algorithms by introducing a high-level graph, induced by the connections of the disjoint subgraphs, as illustrated in Fig. 2. Our distributed algorithm has the following properties: If the fractional entropy approximation in (5) is strictly concave then the algorithm converges for all $\epsilon \geq 0$, and converges to the global optimum when $\epsilon > 0$.

More formally, let $\mathcal{P}$ be a partition of the vertices $\{1, ..., n\}$. Each computer s corresponds to a subgraph G_s induced by the vertices $s \in \mathcal{P}$. Each subgraph G_s describes the marginalization constraints enforced by its computer. Therefore, every computer s deals with its unique set of beliefs $b_i^s(x_i)$ for every $i \in s$ and $b_\alpha^s(\mathbf{x}_\alpha)$ for every $\alpha \in N(i)$. Every computer enforces the marginalization constraints of its corresponding beliefs, *i.e.* $\sum_{x_\alpha \setminus x_i} b_\alpha^s(\mathbf{x}_\alpha) = b_i^s(x_i)$.

A subset α that is shared among the disjoint subgraphs G_s, corresponds to a set of beliefs $b_\alpha^s(\mathbf{x}_\alpha)$, each of them relates to a different computer listed in $N_\mathcal{P}(\alpha)$. These shared beliefs are optimized separately during the distributed executions. Therefore, in general, the distributed step results in different values of $b_\alpha^s(\mathbf{x}_\alpha)$, for every $s \in N_\mathcal{P}(\alpha)$. Since these beliefs originate from a single $b_\alpha(\mathbf{x}_\alpha)$ in (5) we enforce consistency between them. Formally, we construct the multi-graph $G_\mathcal{P}$ from the original graphical model by collapsing the disjoint subgraphs $s \in \mathcal{P}$ to nodes. An edge

α exists in $G_\mathcal{P}$ if it corresponds to an edge α in the original graph. We denote by $N_\mathcal{P}(s)$ the edges in $G_\mathcal{P}$ that contain the vertex s, and by $N_\mathcal{P}(\alpha)$ the vertices in $G_\mathcal{P}$ that appear in the edge α. In order to keep the beliefs $b_\alpha^s(\mathbf{x}_\alpha)$ consistent with respect to the distributed computing units $G_\mathcal{P}$, we add the constraints $b_\alpha^s(\mathbf{x}_\alpha) = b_\alpha(\mathbf{x}_\alpha)$ for every $s \in N_\mathcal{P}(\alpha)$.

To define (5) with respect to the distributed architecture $G_\mathcal{P}$ we need to balance the entropy $H(\mathbf{b}_\alpha)$ and the energy $\theta_\alpha(\mathbf{x}_\alpha)$ for those beliefs that are shared among the different computers. For every $\alpha \in G_\mathcal{P}$ we set $\hat\theta_\alpha(\mathbf{x}_\alpha) = \theta_\alpha(\mathbf{x}_\alpha)/|N_\mathcal{P}(\alpha)|$, and $\hat c_\alpha = c_\alpha/|N_\mathcal{P}(\alpha)|$. For the remaining α we set $\hat\theta_\alpha(\mathbf{x}_\alpha) = \theta_\alpha(\mathbf{x}_\alpha)$ and $\hat c_\alpha = c_\alpha$. Thus we derive an equivalent program to (5) which encodes the distributed architecture:

$$\max \sum_{s \in G_\mathcal{P}} \sum_{\alpha \in G_s, \mathbf{x}_\alpha} b_\alpha^s(\mathbf{x}_\alpha)\hat\theta_\alpha(\mathbf{x}_\alpha) + \sum_{i \in G_s, x_i} b_i^s(x_i)\theta_i(x_i) \quad (6)$$

$$+\epsilon \sum_{s \in G_\mathcal{P}} \left(\sum_{\alpha \in G_s} \hat c_\alpha H(\mathbf{b}_\alpha^s) + \sum_{i \in G_s} c_i H(\mathbf{b}_i^s) \right)$$

subject to:

$$\forall s, i, x_i, \alpha \in N(i), \quad \sum_{\mathbf{x}_\alpha \setminus x_i} b_\alpha^s(\mathbf{x}_\alpha) = b_i^s(x_i)$$

$$\forall s, \alpha \in N_\mathcal{P}(s), \mathbf{x}_\alpha, \quad b_\alpha^s(\mathbf{x}_\alpha) = b_\alpha(\mathbf{x}_\alpha)$$

We would like to use a message-passing algorithm that sends messages along the edges of the graphical model. The graph structure is encoded in the program constraints, therefore the dual program, whose Lagrange multipliers correspond to the primal program constraints, respects the structure of the graph. There are two types of Lagrange multipliers: $\lambda_{i \to \alpha}(x_i)$ enforce the marginalization constraints within each computer, while $\nu_{s \to \alpha}(\mathbf{x}_\alpha)$ enforce the consistency constraints between the different computers.

Claim 1 *Set $\nu_{s \to \alpha} = 0$ for every $\alpha \notin G_\mathcal{P}$. Then the following program is the dual program for the distributed convex belief propagation in (6):*

$$\sum_{s, \alpha \in G_s} \epsilon \hat c_\alpha \ln \sum_{\mathbf{x}_\alpha} \exp \left(\frac{\hat\theta_\alpha(\mathbf{x}_\alpha) + \sum_{i \in N(\alpha) \cap s} \lambda_{i \to \alpha}(x_i) + \nu_{s \to \alpha}(\mathbf{x}_\alpha)}{\epsilon \hat c_\alpha} \right)$$

$$+ \sum_i \epsilon c_i \ln \sum_{x_i} \exp \left(\frac{\theta_i(x_i) - \sum_{\alpha \in N(i)} \lambda_{i \to \alpha}(x_i)}{\epsilon c_i} \right)$$

subject to the constraints $\sum_{s \in N_\mathcal{P}(\alpha)} \nu_{s \to \alpha}(\mathbf{x}_\alpha) = 0$.

Proof: In supplementary material ☐

We perform a block coordinate descent on the dual program. Fixing the consistency messages between computers, $\nu_{s \to \alpha}(\mathbf{x}_\alpha)$, the optimal $\lambda_{i \to \alpha}(x_i)$ is computed for every

Algorithm 1 (Distributed Convex Belief Propagation) *Set* $\hat{\psi}_\alpha(\mathbf{x}_\alpha) = \exp(\hat{\theta}_\alpha(\mathbf{x}_\alpha))$, $\psi_i(x_i) = \exp(\theta_i(x_i))$. *Set* $n_{i\to\alpha}(\mathbf{x}_\alpha) = 1$ *and set* $n_{s\to\alpha}(\mathbf{x}_\alpha) = 1$. *Repeat until convergence:*

1. For $s \in \mathcal{P}$ in parallel: Iterate over $i \in s$

$$\forall x_i \; \forall \alpha \in N(i), \quad m_{\alpha\to i}(x_i) = \left(\sum_{\mathbf{x}_\alpha \setminus x_i} \left(\hat{\psi}_\alpha(\mathbf{x}_\alpha) \prod_{j\in N(\alpha)\cap s\setminus i} n_{j\to\alpha}(x_j) \cdot n_{s\to\alpha}(\mathbf{x}_\alpha) \right)^{1/\epsilon\hat{c}_\alpha} \right)^{\epsilon\hat{c}_\alpha}$$

$$\forall \alpha \in N(i) \; \forall x_i, \quad n_{i\to\alpha}(x_i) \propto \left(\psi_i(x_i) \prod_{\beta\in N(i)} m_{\beta\to i}(x_i) \right)^{\hat{c}_\alpha/\hat{c}_i} \Big/ m_{\alpha\to i}(x_i)$$

2.

$$\forall s \in G_\mathcal{P} \quad \forall \alpha : \; \alpha \text{ is edge in } G_\mathcal{P} \qquad n_{s\to\alpha}(\mathbf{x}_\alpha) = \frac{1}{|N_\mathcal{P}(\alpha)|} \prod_{i\in N(\alpha)} n_{i\to\alpha} \Big/ \prod_{i\in s\cap N(\alpha)} n_{i\to\alpha}(x_i)$$

Figure 3. Our distributed convex belief propagation algorithm extends the sequential convex belief-propagation by adding messages $n_{s\to\alpha}(\mathbf{x}_\alpha)$ to maintain consistency between the distributed executions.

$i \in G_s$ without considering the information in other computers. The consistency messages $\nu_{s\to\alpha}(\mathbf{x}_\alpha)$ are computed in closed-form by synchronizing messages between the different computers.

Claim 2 *For every $s \in G_\mathcal{P}$, set $\mu_{\alpha\to i}(x_i)$ to be*

$$\epsilon\hat{c}_\alpha \ln \sum_{\mathbf{x}_\alpha \setminus x_i} \exp\left(\frac{\hat{\theta}_\alpha(\mathbf{x}_\alpha) + \sum\limits_{j\in N(\alpha)\cap s} \lambda_{j\to\alpha}(x_i) + \nu_{s\to\alpha}(\mathbf{x}_\alpha)}{\epsilon\hat{c}_\alpha} \right)$$

then the block coordinate descent on $\lambda_{i\to\alpha}(x_i)$ takes the form

$$\lambda_{i\to\alpha}(x_i) = \frac{\hat{c}_\alpha}{\hat{c}_i} \left(\theta_i(x_i) + \sum_{\beta\in N(i)} \mu_{\beta\to i}(x_i) \right) - \mu_{\alpha\to i}(x_i),$$

where $\hat{c}_i = c_i + \sum_{\alpha\in N(i)} \hat{c}_\alpha$. The block coordinate descent on $\nu_{s\to\alpha}(\mathbf{x}_\alpha)$ subject to the constraints takes the form:

$$\nu_{s\to\alpha}(\mathbf{x}_\alpha) = \frac{1}{|N_\mathcal{P}(\alpha)|} \sum_{i\in N(\alpha)} \lambda_{i\to\alpha}(x_i) - \sum_{i\in N(\alpha)\cap s} \lambda_{i\to\alpha}(x_i)$$

The block coordinate descent steps above are guaranteed to converge for $\epsilon, c_\alpha, c_i \geq 0$, and guaranteed to reach the optimum of (5, 6) for $\epsilon, c_\alpha, c_i > 0$.

Proof: In supplementary material $\square$

The order of the block coordinate descent updates does not change the convergence guarantees of the algorithm. For computational efficiency, we would like to iteratively update messages within a computer, namely $\lambda_{i\to\alpha}(x_i)$, followed by executing a consistency step message-passing between computers by updating messages $\nu_{s\to\alpha}(\mathbf{x}_\alpha)$. The algorithm is summarized in Fig. 3.

5. Experimental Evaluation

We evaluate our approach in the task of stereo reconstruction. We use truncated linear factor potentials for smoothness and the sum of absolute differences as local evidence. First, we compare our approach to state-of-the-art message passing packages. We then evaluate how often one should send messages between the distributed programs to ensure reasonable convergence rates and low communication overhead. Finally, we demonstrate our method's ability to compute disparity maps from high-resolution images with large number of labels. For our distributed algorithm, we leverage nine 2.4 GHz x64 Quad-Core computers with 24 GB memory each, connected via a standard local area network.

We first compare our approach to publicly available state-of-the-art BP packages using the Tsukuba image pair from the Middleburry data set [22], as this is the most used example of MRFs in the vision literature. According to our knowledge, libDAI 0.2.7 [17] and GraphLAB [16] are the two most well known, up-to-date and publicly available message passing frameworks. We also compare to our convex BP implementation, named "cBP (General)," which is capable of handling arbitrary graphical models with arbitrary potentials. All algorithms run on a single-machine/quad-core computer. Note that the BP algorithm implemented in libDAI is not parallelized for multiple cores and thus it is expected to be slower by a factor of four (i.e., the four cores of the machine). On the other hand, GraphLab has a parallel implementation, and thus we expect it to be faster than libDAI. Another difference between the implementations is that the two baselines, libDAI and GraphLAB, use double precision while we use single precision to reduce the memory requirements when applying the algorithm to large imagery. As many MRFs in computer vision consider only pairwise factors, we also implemented a dedicated version and denote it by "cBP (Ded.)." To be fair in our comparisons we modified the libDAI implementation

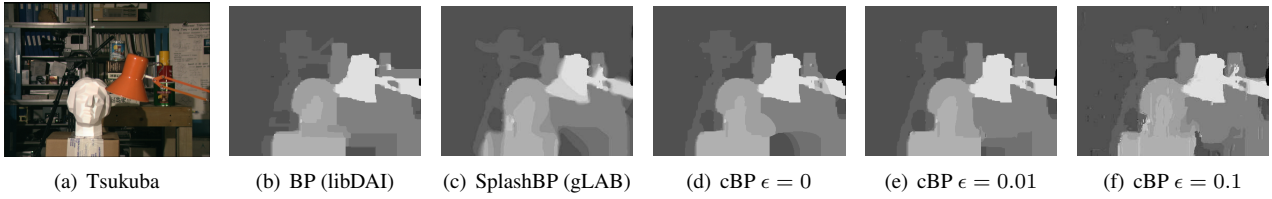| (a) Tsukuba | (b) BP (libDAI) | (c) SplashBP (gLAB) | (d) cBP $\epsilon = 0$ | (e) cBP $\epsilon = 0.01$ | (f) cBP $\epsilon = 0.1$ |

Figure 4. Disparity maps for the Tsukuba image for different algorithms. 1000 Iterations were used for (b) and (d). The splash scheduler, whose result is given in (c) stopped if the change is less than 10^{-10}. The result for $\epsilon = 0.01$ is given in (e) after 100000 iterations. The result in (f) is upon convergence, *i.e.* all constraints are fulfilled.

such that it does not check for convergence.

We compare our approach and the baselines using two measures of efficiency. The first one considers the number of nodes the algorithm updates per second [16]. The second one is the convergence rate, *i.e.* the primal value achieved within a certain number of iterations. Note that we maximize the primal, therefore high numbers are better than lower ones. As shown in Tab. 1, our implementation of convex BP outperforms the baselines in all measures. Importantly, the primal values of libDAI and GraphLab (gLAB) are far worse than the ones returned by convex BP. The relative duality gap ($\frac{dual-primal}{primal}$) of convex BP ($\epsilon = 0$) after 1000 iterations is 0.06%.

We now split the MRF for the Tsukuba image pair into nine equally sized partitions such that computation can be distributed onto the nine machines that form our cluster. We measure the performance of our approach including all communication overhead. This method is referred to as "dis. cBP (Ded.)." As depicted by Tab. 1 our distributed parallel implementation outperforms the baselines in all measures. For a fixed number of iterations the primal energy is expected to be smaller than the one given by the non-distributed convex BP. Note that, however, in this case they are almost identical. Truncated linear potentials admit a more efficient implementation for message computation [2]. We did not implement this specialization, as we intended to investigate a framework that is applicable to more general cases. We nevertheless emphasize that further speedup is possible if required for a particular application. However, the main speedup comes from the number of machines. This is important since large clusters, such as *Amazon EC2* are commonly used these days.

Fig. 4 depicts the disparity maps of libDAI, SplashBP [5] implemented in GraphLAB as well as our approach after 1000 iterations. For SplashBP we specified a bound of 10^{-10} as the stopping criterion. In our experiments, the roundrobin scheduler of GraphLAB did not perform as well as SplashBP. We therefore omit these results.

Next, we show that just splitting an image onto multiple machines without ensuring global consistency (*i.e.*, without passing messages between the splitted graphs) results in artifacts. For faster computation, we downsample the image in Fig. 1 to 0.5 MPixel. Due to the many depth discontinuities this image pair is more challenging than the Tsukuba

Table 1. The runtime (RT) [s], efficiency (Eff.) [Node-updates/μs] and primal energy after 1000 iterations using the Tsukuba data as input. Note that the dedicated implementations cannot be used for a fair comparison and the non-parallelized libDAI is expected to be slower by a factor of four. The given primals were divided by the factor 10^6.

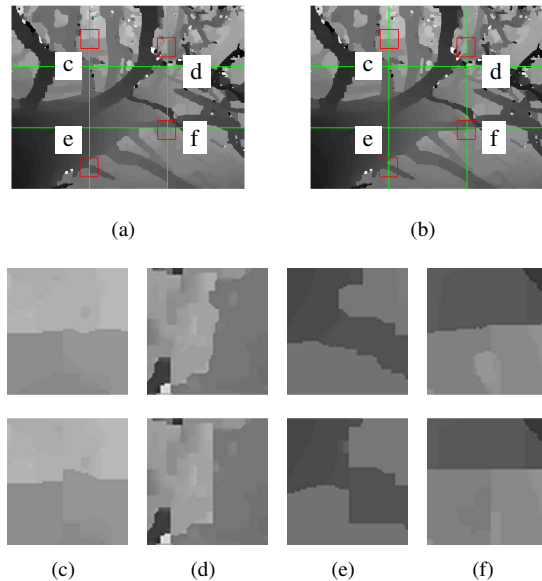| Method	RT [s]	Eff. [nodes/μs]	primal
BP (libDAI)	2617/4	0.04	1.0241
BP (gLAB RR)	1800	0.06	1.0113
SplashBP (gLAB)	689	0.06	1.0121
cBP (General)	371	0.29	**1.0450**
cBP (Ded.)	113	0.94	**1.0450**
dis. cBP (Ded.)	**18**	**5.8**	1.0449

| (a) | (b) |

| (c) | (d) | (e) | (f) |

Figure 5. (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).

pair. The results after 1000 iterations are given in Fig. 5(a) if the individual problems are treated independently, and in Fig. 5(b) when using our approach, *i.e.* transmitting messages between the individual graphs every ten iterations. For the readers convenience we highlighted the differences on the boundaries of the partitions. As expected we observe artifacts on the form of hallucinated discontinuities.
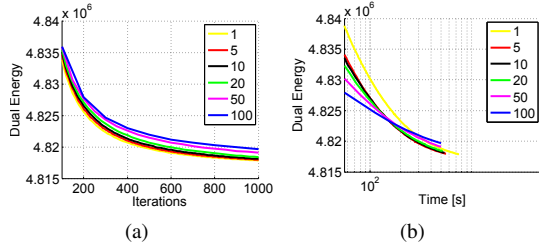
Figure 6. Convergence of the dual energy of the 0.5 MPixel tree image for different update schedules w.r.t. (a) number of iterations and (b) time.

Table 2. The relative duality gap of "cBP (Ded.)" and "dis. cBP (Ded.)" after 1000 iterations. Tasks that were not computed are denoted by *.

MPixel	Labels	dis. cBP (Ded.)	cBP (Ded.)
0.5	58	0.09%	0.09%
1	82	0.13%	0.12%
2	116	0.22%	0.20%
3	142	0.28%	0.26%
4	162	0.30%	0.31%
6	200	0.38%	*
12	283	0.60%	*

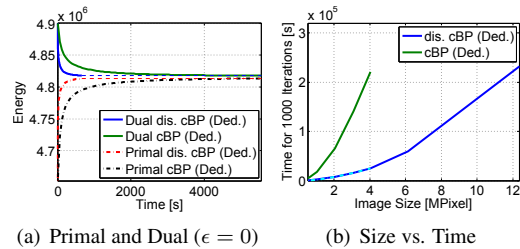(a) Primal and Dual ($\epsilon = 0$)   (b) Size vs. Time

Figure 7. (a) Primal and dual energy for the 0.5 MPixel tree image. (b) Computation time for different image sizes. Note that the number of labels depends on the image size.

We then test how often we are required to update between the individual machines such that we do not degrade the convergence rate. To this end, we compare different schedules for updating the splitted variables, *i.e.* updating every 1, 5, 10, 20, 50 and 100 iterations. The results w.r.t. iterations and time are illustrated in Fig. 6(a) and Fig. 6(b). We conclude that updating every 10 iterations is a reasonable tradeoff between communication overhead and impact on the energy, *i.e.*, convergence rate.

We now demonstrate our method's ability to estimate disparity maps on large scale images with large label sets. To demonstrate the effectiveness of our approach we compare the dedicated implementations of convex BP and our distributed algorithm that uses the nine machines forming our cluster. The local evidence is computed by matching a rectified 6 MPixel stereo pair downloaded from Flickr. To obtain smaller or larger sized images we down- or upsample the stereo pair and adapt the number of labels as given in Tab. 2. We run all BP algorithms for 1000 iterations.

As mentioned previously, the memory requirements of message-passing algorithms are one of their biggest disadvantages. We therefore expect to run out of memory even on a 24 GB machine. Considering a four-connected 6 MPixel image with 200 labels per node we obtain $(199 \cdot 4 \cdot 6 \cdot 10^6)$ variables which results in more than 19.1 GB of float variables. Including the additional memory required to store the graphical model, we were just able to fit everything into 24 GB. The estimated time to finish 1000 iterations is around 9 days; 12 MPixel images would clearly exceed the available memory space.

Dividing the 6 MPixel image into nine equally sized blocks requires around 2 GB of memory to store all the messages. Neglecting the time required for communication we also benefit from the nine times higher computational power. Our approach took 16.6 hours to perform 1000 iterations including computation of the dual value as well as the communication overhead of transferring information between the sub-problems every ten iterations. The resulting relative duality gap (DG) is given for varying image sizes and adapted number of labels in Tab. 2. As expected, the relative DG increases with the image size. Even

though we transfer information every 10 iterations only, the DG differs only slightly.

Fig. 7(a) depicts primal and dual energy as a function of time. The time necessary to perform 1000 iterations as a function of the image size is shown in Fig. 7(b). This shows that we obviously did not change the complexity of the belief propagation algorithm, which is $O(nk^2)$, with n denoting the number of random variables in the graphical model and k the number of labels per variable, when considering pairwise factors. Nevertheless we are now capable of reducing the runtime and memory consumptions as the graph to be solved in each machine is smaller. The complexity is then $O\left(\frac{nk^2}{|\mathcal{P}|}\right)$, with $|\mathcal{P}|$ the number of machines.

As mentioned previously, convergence to a global optimum of (6) is not guaranteed for $\epsilon = 0$. If an application requires those guarantees, we can approximate the $\epsilon = 0$ with $\epsilon \to 0$ and conserve the guarantees. We provide two examples for $\epsilon = 0.01$ and $\epsilon = 0.1$ in Fig. 4(e) and Fig. 4(f) for completeness. Note, that unless special lookup tables are used, computation time will increase as the max-function is replaced with log and exp. The corresponding energy curves for $\epsilon = 0.01$ and $\epsilon = 0.1$ are given in Fig. 8(a) and Fig. 8(b) respectively. For $\epsilon = 0$, 1000 iterations are sufficient for a small relative DG. Note, that this contrasts the case $\epsilon > 0$, were we generally require much more iterations.
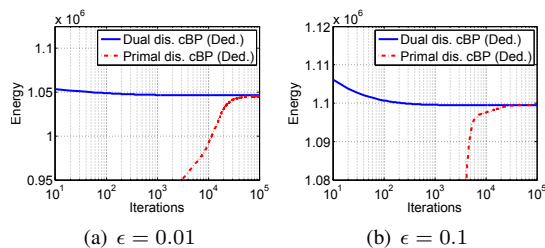
	(a) $\epsilon = 0.01$	(b) $\epsilon = 0.1$

Figure 8. (a) Primal and (b) dual energy obtained from "dis. cBP (Ded.)" when approximating the MAP problem with $\epsilon = 0.01$ and $\epsilon = 0.1$ respectively. The corresponding disparity maps are given in Fig. 4(e) and Fig. 4(f).

6. Conclusion and Future Work

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 into multiple machines. Importantly the convergence and optimality guarantees 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. The main benefit of our approach comes from the use of multiple computers. Thus we expect that running our algorithm within large clusters, such as *Amazon EC2*, will result in orders of magnitude further speed up.

References

[1] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *PAMI*, 2001. 1, 2

[2] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. *IJCV*, 70(1):41–54, 2006. 1, 2, 6

[3] W. T. Freeman, T. R. Jones, and E. C. Pasztor. Example-based super-resolution. *IEEE Comput. Graph. Appl.*, 22(2):56–65, 2002. 1

[4] A. Globerson and T. S. Jaakkola. Fixing max-product: convergent message passing algorithms for MAP relaxations. In *NIPS*, 2007. 1, 3

[5] J. E. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. In *AISTATS*, pages 177–184, 2009. 6

[6] T. Hazan and A. Shashua. Norm-Product Belief Propagation: Primal-Dual Message-Passing for Approximate Inference. *Arxiv preprint arXiv:0903.3127*, 2009. 1, 2

[7] T. Heskes. Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies. *Journal of Artificial Intelligence Research*, 26(1):153–190, 2006. 1, 3

[8] P. Kohli, L. Ladickỳ, and P. H. S. Torr. Robust higher order potentials for enforcing label consistency. *IJCV*, 82(3):302–324, 2009. 2

[9] V. Kolmogorov and R. Zabih. What energy functions can be minimizedvia graph cuts? *PAMI*, pages 147–159, 2004. 2

[10] N. Komodakis, N. Paragios, and G. Tziritas. MRF Energy Minimization & Beyond via Dual Decomposition. *PAMI*, 2010. 1, 3

[11] F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *Trans. on Information Theory*, 47(2):498–519, 2001. 2

[12] X. Lan, S. Roth, D. Huttenlocher, and M. Black. Efficient belief propagation with learned higher-order Markov random fields. In *ECCV*, pages 269–282, 2006. 1

[13] V. Lempitsky, C. Rother, and A. Blake. Logcut-efficient graph cut optimization for markov random fields. In *ICCV*, pages 1–8, 2007. 2

[14] A. Levin and Y. Weiss. Learning to Combine Bottom-Up and Top-Down Segmentation. In *ECCV*, 2006. 1

[15] A. Levin, A. Zomet, and Y. Weiss. Learning how to inpaint from global image statistics. In *ICCV*, pages 305–312, 2003. 1

[16] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. Hellerstein. Graphlab: A new parallel framework for machine learning. In *UAI*, 2010. 2, 5, 6

[17] J. Mooij. libDAI: A free/open source C++ library for discrete approximate inference methods, 2009. http://www.libdai.org. 5

[18] J. Mooij and H. Kappen. Sufficient conditions for convergence of loopy belief propagation. In *UAI*, 2005. 2

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988. 1, 2

[20] Y. Pritch, E. Kav-Venaki, and S. Peleg. Shift-map image editing. In *ICCV*, pages 151–158, 2010. 2

[21] R. Rockafellar. *Convex analysis*. Princeton university press, 1970. 3

[22] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1/2/3):7–42, 2002. 5

[23] S. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68(2):399–410, 1994. 3

[24] P. Strandmark and F. Kahl. Parallel and distributed graph cuts by dual decomposition. In *CVPR*, pages 2085–2092, 2010. 2

[25] J. Sun, N. Zheng, and H. Shum. Stereo matching using belief propagation. *PAMI*, pages 787–800, 2003. 1

[26] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. *Trans. on Information Theory*, 51(7):2313–2335, 2005. 3

[27] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. MAP estimation via agreement on trees: message-passing and linear programming. *Trans. on Information Theory*, 51(11):3697–3717, 2005. 3

[28] T. Werner. A linear programming approach to max-sum problem: A review. *PAMI*, 29(7):1165–1179, 2007. 1, 3

[29] C. Yanover, T. Meltzer, and Y. Weiss. Linear Programming Relaxations and Belief Propagation–An Empirical Study. *JMLR*, 7:1907, 2006. 3