MPLP++: Fast, Parallel Dual Block-Coordinate Ascent for Dense Graphical Models

Siddharth Tourani[1(✉)], Alexander Shekhovtsov[2], Carsten Rother[1], and Bogdan Savchynskyy[1]

[1] Visual Learning Lab, IWR, University of Heidelberg, Heidelberg, Germany
{siddharth.tourani,Carsten.Rother,bogdan.savchynskyy}@iwr.uni-heidelberg.de
[2] Centre for Machine Perception, Czech Technical University, Prague, Czech Republic
shekhovtsov@gmail.com

Abstract. Dense, discrete Graphical Models with pairwise potentials are a powerful class of models which are employed in state-of-the-art computer vision and bio-imaging applications. This work introduces a new MAP-solver, based on the popular Dual Block-Coordinate Ascent principle. Surprisingly, by making a small change to a low-performing solver, the Max Product Linear Programming (MPLP) algorithm [7], we derive the new solver MPLP++ that significantly outperforms all existing solvers by a large margin, including the state-of-the-art solver Tree-Reweighted Sequential (TRW-S) message-passing algorithm [17]. Additionally, our solver is highly parallel, in contrast to TRW-S, which gives a further boost in performance with the proposed GPU and multi-thread CPU implementations. We verify the superiority of our algorithm on dense problems from publicly available benchmarks as well as a new benchmark for 6D Object Pose estimation. We also provide an ablation study with respect to graph density.

Keywords: Graphical models · Block-Coordinate-Ascent · Message passing algorithms

1 Introduction

Undirected discrete graphical models with dense neighbourhood structure are known to be much more expressive than their sparse counterparts. A striking example is the fully-connected Conditional Random Field (CRF) model with Gaussian pairwise potentials [23], significantly improving the image segmentation field, once an efficient solver for the model was proposed. More recently,

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-01225-0_16) contains supplementary material, which is available to authorized users.

V. Ferrari et al. (Eds.): ECCV 2018, LNCS 11208, pp. 264–281, 2018.
https://doi.org/10.1007/978-3-030-01225-0_16

various applications in computer vision and bio-imaging have successfully used fully-connected or densely-connected, pairwise models with non-Gaussian potentials. Non-Gaussian potentials naturally arise from application-specific modelling or the necessity of robust potentials. A prominent application of the non-Gaussian fully-connected CRF case achieved state-of-the-art performance in 6D object pose estimation [29], with an efficient *application-specific* solver. Other examples of densely connected models were proposed in the area of stereo-reconstruction [19], body pose estimation [2,16,30], bio-informatics [12] etc.

An efficient solver is a key condition to make such expressive models efficient in practice. This work introduces such a solver, which outperforms all existing methods for a class of dense and semi-dense problems with non-Gaussian potentials. This includes Tree-Reweighted Sequential (TRW-S) message passing, which is typically used for general pairwise models. We would like to emphasize that efficient solvers for this class are highly desirable, even in the age of deep learning. The main reason is that such expressive graphical models can encode information which is often hard to learn from data, since very large training datasets are needed to learn the application specific prior knowledge. In the above mentioned 6D object pose estimation task, the pairwise potentials encode length-consistency between the observed data and the known 3D model and the unary potentials are learned from data. Other forms of combining graphical models with CNNs such as Deep-Structured-Models [3] can also benefit from the proposed solver.

Linear Programming (LP) relaxation is a powerful technique that can solve exactly all known tractable maximum a posteriori (MAP) inference problems for undirected graphical models (those known to be polynomially solvable) [20]. Although there are multiple algorithms addressing MAP inference, which we discuss in Sect. 2, the linear programs obtained by relaxing the MAP-inference problem are not any simpler than general linear programs [31]. This implies that algorithms solving it exactly are bounded by the computational complexity of the general LP and do not scale well to problems of large size. Since LP relaxations need not be tight, solving it optimally is often impractical. On the other hand, block coordinate ascent (BCA) algorithms for the LP dual problem form an approach delivering fast and practically useful approximate solutions. TRW-S [17] is probably the most well-known and efficient solver of this class, as shown in [14]. Since due to the graph density the model size grows quadratically with the number of variables, a scalable solver must inevitably be highly parallelizable to be of practical use. Our work improves another well-known BCA algorithm of this type, MPLP [7] (Max Product Linear Programming algorithm) and proposes a parallel implementation as explained next.

Contribution. We present a new state-of-the-art parallel BCA algorithm of the same structure as MPLP, *i.e.* an elementary step of our algorithm updates all dual variables related to a single graph edge. We explore the space of such edge-wise updates and propose that a different update rule inspired by [43] can be employed, which significantly improves the practical performance of MPLP. Our method with the new update is termed MPLP++. The difference in the updates

stems from the fact that the optimization in the selected block of variables is non-unique and the way the update utilizes the degrees of freedom to which the block objective is not sensitive significantly affects subsequent updates and thus the whole performance.

We propose the following theoretical analysis. We show that MPLP++ converges towards arc-consistency, similarly to the convergence result of [36] for min-sum diffusion. We further show that given any starting point, an iteration of the MPLP++ algorithm, which processes all edges of the graph in a specified order always results in a better objective than the same iteration of MPLP. For multiple iterations this is not theoretically guaranteed, but empirically observed in all our test cases. All proofs relating to the main paper are given in the supplement.

Another important aspect that we address is parallelization. TRW-S is known as a "sequential" algorithm. However, it admits parallelization especially for bipartite graphs [17], which is exploited in specialized implementations [5,9] for 4-connected grid graphs. The parallelization there gives a speed-up factor of $O(n)$, where n is the number of nodes in the graph. A parallel implementation for dense graphs has not been proposed. We observe that in MPLP a group of non-incident edges can be updated in parallel. We pre-compute a schedule maximizing the number of edges that can be processed in parallel using an exact or a greedy maximum matching. The obtainable theoretical speed-up is at least $n/2$ for *any* graph, including dense ones. We consider two parallel implementations, suitable for CPU and GPU architectures respectively. A further speed-up is possible by utilizing parallel algorithms for lower envelopes in message passing (see Sect. 4 and [4]).

The new MPLP++ method consistently outperforms all its competitors, including TRW-S [17], in the case of densely (not necessarily fully) connected graphs, even in the sequential setting: In our experiments it is 2 to 10 times faster than TRW-S and 5 to 150 times faster than MPLP depending on the dataset and the required solution precision. The empirical comparison is conducted on several datasets. As there are only few publicly available ones, we have created a new dataset related to the 6D pose estimation problem [29].

2 Related Work

The general MAP inference problem for discrete graphical models (formally defined in Sect. 3) is NP-hard and is also hard to approximate [24]. A natural linear programming relaxation is obtained by formulating it as a 0-1 integer linear program (ILP) and relaxing the integrality constraints [38] (see also the recent review [47]). A hierarchy of relaxations is known [49], from which the so-called Base LP relaxation, also considered in our work, is the simplest one. It was shown [20,46] that this relaxation is tight for all tractable subclasses of the problem. For many other classes of problems it provides approximation guarantees, reviewed in [24].

Apart from general LP solvers, a number of specialized algorithms exist that take advantage of the problem structure and guarantee convergence to an optimal solution of the LP relaxation. This includes proximal [26,27,33,40], dual subgradient [21,37,44], bundle [15], mirror-descent [25] smoothing-based [28,34,35] and (quasi-) Newton [13] methods.

However, as it was shown in [32], the linear programs arising from the relaxation of the MAP inference problem have the same computational complexity as general LPs. At the same time, if the problem is hard, solving the relaxation to optimality may be of low practical utility. A comparative study [14] notes that TRW-S [17] is the most efficient solver for the relaxation in practice. It belongs to the class of BCA methods for the LP dual that includes also MPLP [7], min-sum diffusion [22,36] and DualMM [43] algorithms. BCA methods are not guaranteed to solve the LP dual to optimality. They may get stuck in a suboptimal point and be unable to compute primal LP solutions unless integer solutions are found (which is not always the case). However, they scale extremely well, take advantage of fast dynamic programming techniques, solve exactly all submodular problems [39] and provide good approximate solutions in general vision benchmarks.

3 Preliminaries

Notation. $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denotes an undirected graph, with vertex set $\mathcal{V}$ (we assume $\mathcal{V} = \{1, \dots, n\}$) and edge set $\mathcal{E}$. The notation $uv \in \mathcal{E}$ will mean that $\{u, v\} \in \mathcal{E}$ and $u < v$ with respect to the order of $\mathcal{V}$. Each node $u \in \mathcal{V}$ is associated with a label from a finite *set of labels* $\mathcal{Y}$ (for brevity w.l.o.g. we will assume that it is the same set for all nodes). The label space for a pair of nodes $uv \in \mathcal{E}$ is $\mathcal{Y}^2$ and for all nodes it is $\mathcal{Y}^{\mathcal{V}}$.

For each node and edge the *unary* $\theta_u : \mathcal{Y} \to \mathbb{R}$, $u \in \mathcal{V}$ and *pairwise* cost functions $\theta_{uv} : \mathcal{Y}^2 \to \mathbb{R}$, $uv \in \mathcal{E}$, assign a cost to a label or label pair, respectively. Let $\mathcal{I} = (\mathcal{V} \times \mathcal{Y}) \cup (\mathcal{E} \times \mathcal{Y}^2)$ be the index set enumerating all labels and label pairs in neighbouring graph nodes. Let the *cost vector* $\theta \in \mathbb{R}^{\mathcal{I}}$ contain all values of the functions θ_u and θ_{uv} as its coordinates.

The *MAP-inference* problem for the graphical model defined by the triple $(\mathcal{G}, \mathcal{Y}^{\mathcal{V}}, \theta)$ consists in finding the labelling with the smallest total cost, *i.e.*:

$$y^* = \arg\min_{y \in \mathcal{Y}^{\mathcal{V}}} \left[E(y|\theta) := \sum_{v \in \mathcal{V}} \theta_v(y_v) + \sum_{uv \in \mathcal{E}} \theta_{uv}(y_{uv}) \right]. \tag{1}$$

This problem is also known as *energy minimization* for graphical models and is closely related to weighted and valued constraint satisfaction problems. The total cost E is also often called *energy*.

The problem (1) is in general NP-hard and is also hard to approximate [24]. A number of approaches to tackle it in different practical scenarios are reviewed in [10,14]. One of the widely applicable techniques is based on (approximately) solving its linear programming (LP) relaxation as discussed in Sect. 2.

Dual Problem. Most existing solvers for the LP relaxation tackle its dual form, which we introduce now. It is based on the fact that the representation of the energy function $E(y|\theta)$ using unary θ_u and pairwise θ_{uv} costs is not unique. There exist other costs $\hat{\theta} \in \mathbb{R}^{\mathcal{I}}$ such that $E(y|\hat{\theta}) = E(y|\theta)$ for all labelings $y \in \mathcal{Y}^{\mathcal{V}}$.

It is known (see e.g. [47]) and straightforward to check that such *equivalent* costs can be obtained with an arbitrary vector $\phi := (\phi_{v\to u}(s) \in \mathbb{R} \mid u \in \mathcal{V},\ v \in \mathrm{Nb}(u),\ s \in \mathcal{Y})$, where $\mathrm{Nb}(u)$ is the set of neighbours of u in $\mathcal{G}$, as follows:

$$\begin{aligned} \hat{\theta}_u(s) \equiv \theta^{\phi}_u(s) &:= \theta_u(s) + \sum\nolimits_{v\in \mathrm{Nb}(u)} \phi_{v\to u}(s) \\ \hat{\theta}_{uv}(s,t) \equiv \theta^{\phi}_{uv}(s,t) &:= \theta_{uv}(s,t) - \phi_{v\to u}(s) - \phi_{u\to v}(t)\,. \end{aligned} \tag{2}$$

The cost vector θ^{ϕ} is called *reparametrized* and the vector ϕ is known as *reparametrization*. Costs related by (2) are also called *equivalent*. Other established terms for reparametrization are *equivalence preserving* [6] or *equivalent transformations* [38].

By swapping min and $\sum$ operations in (1) one obtains a lower bound on the energy $D(\theta^{\phi}) \le E(y|\theta)$ for all y, which reads

$$D(\theta^{\phi}) := \sum_{u\in\mathcal{V}} \min_{s\in\mathcal{Y}} \theta^{\phi}_u(s) + \sum_{uv\in\mathcal{E}} \min_{(s,t)\in\mathcal{Y}^2} \theta^{\phi}_{uv}(s,t)\,. \tag{3}$$

Although the energy $E(y|\theta)$ remains the same for all equivalent cost vectors (i.e. $E(y|\theta) = E(y|\theta^{\phi})$), the lower bound $D(\theta^{\phi})$ depends on the reparametrization, which is $D(\theta) \ne D(\theta^{\phi})$. Therefore, a natural maximization problem arises as maximization of the lower bound over all equivalent costs: $\max_{\phi} D(\theta^{\phi})$. It is known (see [47]) that this maximization problem can be seen as a dual formulation of the LP relaxation of (1). We will write $D(\phi)$ to denote $D(\theta^{\phi})$, since the cost vector θ is clear from the context. The function $D(\phi)$ is concave, piecewise linear and therefore, non-smooth. In many applications the dimensionality of ϕ often exceeds 10^5 to 10^6 and the respective dual problem $\max_{\phi} D(\phi)$ is large scale.

4 Dual Block-Coordinate Ascent

As we discussed in Sect. 2, BCA methods, although not guaranteed to solve the dual to the optimality, provide good solutions for many practical instances and scale very well to large problems. The fastest such methods are represented by methods working with chain subproblems [17,43] or their generalizations [18].

The TRW-S algorithm can be seen as updating a block of dual variables $(\phi_{v\to u},\ v \in \mathrm{Nb}(u))$ "attached" to a node u during each elementary step. The same block of variables is also used in the *min-sum diffusion* algorithm [36], as well as in *convex message passing* [8], and [18] gives a generalization of such methods.

However, the update coefficients in TRW-S are related to the density of the graph and its advantage diminishes when the graph becomes dense. We show

that for dense graphs updating a block of dual variables $\phi_{u\leftrightarrow v} = (\phi_{v\to u}, \phi_{u\to v})$ associated to an edge $uv \in \mathcal{E}$ can be more efficient. Such updates were previously used in the MPLP algorithm [7]. We show that our MPLP++ updates differ in detail but bring a significant improvement in performance.

Block Optimality Condition. For further analysis of BCA algorithms we will require a sufficient condition of optimality w.r.t. the selected block of dual variables. The restriction of the dual to the block of variables $\phi_{u\leftrightarrow v}$ is given by the function:

$$D_{uv}(\phi_{u\leftrightarrow v}) := \min_{(s,t)\in\mathcal{Y}^2} \theta^{\phi}_{uv}(s,t) + \min_{s\in\mathcal{Y}} \theta^{\phi}_{u}(s) + \min_{t\in\mathcal{Y}} \theta^{\phi}_{v}(t)\,, \tag{4}$$

Maximizing D_{uv} is equivalent to performing a BCA w.r.t. $\phi_{u\leftrightarrow v}$ for $D(\phi)$. The necessary and sufficient condition of a maximum of D_{uv} are given by the following.

Proposition 1. *Reparametrization $\phi_{u\leftrightarrow v}$ maximizes $D_{uv}(\cdot)$ iff there exist $(s,t) \in \mathcal{Y}^2$ such that s minimizes $\theta^{\phi}_{u}(\cdot)$, t minimizes $\theta^{\phi}_{v}(\cdot)$ and (s,t) minimizes $\theta^{\phi}_{uv}(\cdot,\cdot)$.*

This condition is trivial to check, it is a special case of arc consistency [47] or weak tree-agreement [17] when considering a simple graph with one edge. It is also clear that $\phi_{u\leftrightarrow v}$ satisfying this condition is not unique. For BCA algorithms it means that there are degrees of freedom in the block which do not directly affect the objective. By moving on a plateau, they can nevertheless affect subsequent BCA updates. Therefore, the performance of the algorithm will be very much dependent on the particular BCA update rule satisfying Proposition 1.

Block Coordinate Ascent Updates. Given the form of the restricted dual (4) on the edge uv, all BCA-updates can be described as follows. Assume θ is the current reparametrized cost vector, *i.e.* $\theta = \bar{\theta}^{\bar{\phi}}$ for some initial $\bar{\theta}$ and the current reparametrization $\bar{\phi}$.

Definition 1. *A* BCA update *takes as the input an edge $uv \in \mathcal{E}$ and costs $\theta_{uv}(s,t)$, $\theta_u(s)$ and $\theta_v(t)$ and outputs a reparametrization $\phi_{u\leftrightarrow v}$ satisfying Proposition 1. W.l.o.g., we assume that it will also satisfy $\min_{(s,t)} \theta^{\phi}_{uv}(s,t) = 0$.*[1]

According to (2) a BCA-update results in the following reparametrized potentials:

$$\theta^{\phi}_{u} = \theta_u + \phi_{v\to u}, \quad \theta^{\phi}_{v} = \theta_v + \phi_{u\to v}, \quad , \theta^{\phi}_{uv}(s,t) = \theta_{uv} - \phi_{v\to u} - \phi_{u\to v}. \tag{5}$$

Note that since all reparametrizations constitute a vector space, after a BCA-update ϕ we can update the current total reparametrization as $\bar{\phi} := \bar{\phi} + \phi$.

[1] This fixes the ambiguity w.r.t. a constant that can be otherwise added to the edge potential and subtracted from one of the unary potentials. This constant does not affect the performance of algorithms.

We will consider BCA-updates of the following form: first construct the aggregated cost $g_{uv}(s,t) = \theta_{uv}(s,t) + \theta_u(s) + \theta_v(t)$. This corresponds to applying a reparametrization $\mathring{\phi}_{u \leftrightarrow v} = (-\theta_u, -\theta_v)$, which gives $\theta^{\mathring{\phi}}_{uv} = g_{uv}$, $\theta^{\mathring{\phi}}_u = \theta^{\mathring{\phi}}_v = 0$. After that, a BCA update forms a new reparametrization $\phi_{u \leftrightarrow v}$ such that $\theta^{\mathring{\phi}+\phi}$ satisfies Proposition 1.

Such BCA-updates can be represented in the following form, which will be simpler for defining and analysing the algorithms.

Definition 2. *Consider a BCA-update using a composite reparametrization* $\mathring{\phi}_{u \leftrightarrow v} + \phi_{u \leftrightarrow v}$. *It can then be fully described by the* reparametrization mapping $\gamma\colon g_{uv} \to (\theta^\gamma_u, \theta^\gamma_v)$, *where* $g_{uv} \in \mathbb{R}^{\mathcal{Y}_{uv}}$, $\theta^\gamma_u = \phi_{v \to u}$ *and* $\theta^\gamma_v = \phi_{u \to v}$.

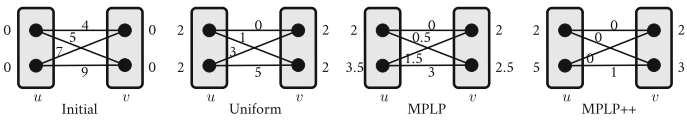


Fig. 1. Illustration of the considered BCA-updates. The gray boxes in the figure represent graph nodes. The black dots in them are the labels. The edges connecting the black dots make up the pairwise costs. The numbers adjacent to the edges and labels are the pairwise and unary costs, respectively.

By construction, θ^γ_u matches the reparametrized unary term $\theta^{\mathring{\phi}+\phi}_u$, θ^γ_v is alike and the reparametrized pairwise term is given by $\theta^\gamma_{uv} = g_{uv} - \phi_{v \to u} - \phi_{u \to v} = g_{uv} - \theta^\gamma_u - \theta^\gamma_v$. In what follows, BCA-update will mean specifically the reparametrization mapping γ. We define now several BCA-updates that will be studied further.

• The `uniform` BCA-update is given by the following reparametrization mapping $\mathcal{U}$:

$$\theta^{\mathcal{U}}_u(s) = \theta^{\mathcal{U}}_v(t) := \tfrac{1}{2} \min_{(s',t') \in \mathcal{Y}^2} g_{uv}(s',t'),\ \forall s', t' \in \mathcal{Y}. \qquad (\mathcal{U})$$

This is indeed just an example, to illustrate the problem of non-uniqueness of the minimizer. It is easy to see that this update satisfies Proposition 1 since both $\theta^{\mathcal{U}}_u(\cdot)$ and $\theta^{\mathcal{U}}_v(\cdot)$ are constant and therefore any pairwise minimizer of $\theta^{\mathcal{U}}_{uv}$ is consistent with them.

• The `MPLP` BCA-update is given by the following reparametrization mapping $\mathcal{M}$:

$$\begin{aligned} \theta^{\mathcal{M}}_u(s) &:= \tfrac{1}{2} \min_{t \in \mathcal{Y}} g_{uv}(s,t),\ \forall s \in \mathcal{Y}, \\ \theta^{\mathcal{M}}_v(t) &:= \tfrac{1}{2} \min_{s \in \mathcal{Y}} g_{uv}(s,t),\ \forall t \in \mathcal{Y}. \end{aligned} \qquad (\mathcal{M})$$

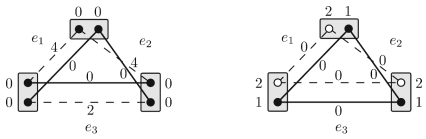
The MPLP algorithm [7] can now be described as performing iterations by applying BCA-update $\mathcal{M}$ to all edges of the graph in a sequence.

• The new MPLP++ BCA-update, that we propose, is based on the *handshake* operation [43]. It is given by the following procedure defining the reparametrization mapping $\mathcal{H}$:

$$\begin{aligned}
&\theta_u^{\mathcal{H}}(s) := \theta_u^{\mathcal{M}}(s), \quad \theta_v^{\mathcal{H}}(s) := \theta_v^{\mathcal{M}}(s), \ \forall s \in \mathcal{Y}\,, \\
&\theta_v^{\mathcal{H}}(t) := \theta_v^{\mathcal{H}}(t) + \min_{s\in\mathcal{Y}}[g_{uv}(s,t) - \theta_v^{\mathcal{H}}(t) - \theta_u^{\mathcal{H}}(s)], \ \forall t \in \mathcal{Y}\,, \qquad (\mathcal{H}) \\
&\theta_u^{\mathcal{H}}(s) := \theta_u^{\mathcal{H}}(s) + \min_{t\in\mathcal{Y}}[g_{uv}(s,t) - \theta_v^{\mathcal{H}}(t) - \theta_u^{\mathcal{H}}(s)], \ \forall s \in \mathcal{Y}\,.
\end{aligned}$$

In other words, the MPLP++ update first performs the MPLP update and then pushes as much cost from the pairwise factor to the unary ones as needed to fulfill $\min_t \theta_{uv}^{\mathcal{M}}(s,t) = \min_s \theta_{uv}^{\mathcal{M}}(s,t) = 0$ for all labels s and t in the nodes u and v respectively. It is also easy to see that the assignment $\theta_v^{\mathcal{H}}(s) := \theta_v^{\mathcal{M}}(s)$ together with the second line in $(\mathcal{H})$ can be equivalently substituted by $\theta_v^{\mathcal{H}}(t) := \min_{s\in\mathcal{Y}}[g_{uv}(s,t) - \theta_u^{\mathcal{H}}(s)], \ \forall t \in \mathcal{Y}$. This allows to perform the MPLP++ update with 3 minimizations over $\mathcal{Y}^2$ instead of 4. Figure 1 shows the result of applying the three BCA-updates on a simple two-node graph.

It is straightforward to show that $\mathcal{M}$ and $\mathcal{H}$ also satisfy Definition 1 (see supplement) and therefore are liable BCA-updates. In spite of that, the behavior of all three updates is notably different, as it is shown by the example in Fig. 2. Therefore, proving only that some algorithm is a BCA, does not imply its efficiency.



(a) Uniform (b) MPLP

Fig. 2. Choosing the right BCA-update is important. Notation has the same meaning as in Fig. 1. Solid lines correspond to the locally optimal pairwise costs connected to the locally optimal labels. Omitted lines in pairwise interactions denote infinite pairwise costs. e_i denotes edge indexes, edge processing order is according to the subscript i. **(a)** Uniform update gets stuck and is unable to optimize the dual further. **(b)** MPLP and MPLP++ attain the dual optimum in one iteration.

Message Passing. Importantly for performance, updates $\mathcal{U}$, $\mathcal{M}$ and $\mathcal{H}$ can be computed using a subroutine computing $\min_t[\theta_{uv}(s,t) + a(t)]$ for all s, where θ is a fixed initial pairwise potential and a is an arbitrary input unary function. This operation occurring in dynamic programming and all of the discussed BCA algorithms, is known as *message passing*. In many cases of practical interest it can be implemented in time $O(|\mathcal{Y}|)$ (e.g. for Potts, absolute difference and quadratic costs) rather than $O(|\mathcal{Y}|^2)$ in the general case, using efficient sequential algorithms, *e.g.*, [1].

Primal Rounding. BCA-algorithms iterating BCA-updates give only a lower bound to the MAP-inference problem (1). To obtain a primal solution, we use a sequential rounding procedure similar to the one proposed in [17]. Assuming we have already computed a primal integer solution x_u^* for all $u < v$, we want to compute x_v^*. To do so, we use the following equation for the assignment

$$x_v^* \in \arg\min_{x_u \in \mathcal{Y}} \left[\theta_v(x_v) + \sum_{u<v|uv\in\mathcal{E}} \theta_{uv}(x_u^*, x_v)\right], \tag{6}$$

where θ is the reparametrized potential produced by the algorithm.

5 Theoretical Analysis

As we prove below, MPLP++ in the limit guarantees to fulfill a necessary optimality condition related to *arc-consistency* [47] and *weak tree-agreement* [17].

Arc-Consistency. Let $[\![\cdot]\!]$ be the Iverson bracket, *i.e.* $[\![A]\!] = 1$ if A holds. Otherwise $[\![A]\!] = 0$. Let $\bar{\theta}_u(s) := [\![\theta_u(s) = \min_{s'} \theta_u(s')]\!]$ and $\bar{\theta}_{uv}(s,t) := [\![\theta_{uv}(s,t) = \min_{s',t'} \theta_{uv}(s',t')]\!]$ be binary vectors with values 1 assigned to the locally minimal labels and label pairs. Let also logical *and* and *or* operations be denoted as $\wedge$ and $\vee$. To the binary vectors they apply coordinate-wise.

Definition 3. *A vector $\bar{\theta} \in \{0,1\}^{\mathcal{I}}$ is called* arc-consistent*, if $\bigvee_{t\in\mathcal{Y}} \bar{\theta}_{uv}(s,t) = \bar{\theta}_u(s)$ for all $\{u,v\} \in \mathcal{E}$ and $s \in \mathcal{Y}$.*

However, arc-consistency itself is not necessary for dual optimality. The necessary condition is existence of node-edge agreement, which is a special case of weak tree agreement [17] when individual nodes and edges are considered as the trees in a problem decomposition. This condition is also known as a non-empty *kernel* [47]/*arc-consistent closure* [6] of the cost vector θ.

Definition 4. *We will say that the costs $\theta \in \mathbb{R}^{\mathcal{I}}$ fulfill* node-edge agreement*, if there is an arc-consistent vector $\xi \in \{0,1\}^{\mathcal{I}}$ such that $\xi \wedge \bar{\theta} = \xi$.*

Convergence of MPLP++. It is clear that all BCA algorithms are monotonous and converge in the dual objective value as soon as the dual is bounded (*i.e.*, the primal is feasible). However, this is a weaker convergence than the desired *node-edge agreement*. To this end we were able to show something in between the two: the convergence of MPLP++ in a measure quantifying violation of the node-edge agreement, a result analogous to [36].

Theorem 1. *The* MPLP++ *algorithm converges to node-edge agreement.*

When comparing different algorithms, the ultimate goal is to prove faster convergence of one compared to the other. We cannot show that the new MPLP++ algorithm has a better theoretical convergence rate. First, such rates are generally unknown for BCA algorithms for non-smooth functions. Second, the considered algorithms are all of the same family and it is likely that their asymptotic

rates are the same. Instead, we study the *dominance*, a condition that allows to rule out BCA updates which are always inferior to others. Towards this end we show that given the same starting dual point, one iteration of `MPLP++` always results in a better objective value than that of `MPLP` and `uniform` BCA. While this argument does not extend theoretically to multiple iterations of each method, we show that it is still true in practice for all used datasets and a significant speed-up (up to two orders) is observed. The experimental comparison in Sect. 7 gives results in wall-clock time as well as in a machine-independent count of the message passing updates performed.

5.1 Analysis of BCA-updates

Definition 5. *A* BCA-iteration α *is defined by the BCA-update* γ *applied to all edges* $\mathcal{E}$ *in some chosen order. Let also* $D(\alpha)$ *represent the dual objective value with the reparametrization defined by the iteration* α *on the input costs* θ*.*

We will analyze BCA-iterations of different BCA-updates w.r.t. the same sequence of edges. The goal is to show that an iteration of the new update dominates the baselines in the dual objective. This property is formally captured by the following definition.

Definition 6. *We will say that a BCA-iteration* α dominates *a BCA-iteration* β*, if for any input costs it holds* $D(\alpha) \geq D(\beta)$*.*

In order to show it, we introduce now and prove later the dominance relations of individual BCA-updates. They are defined not on the dual objective but on all unary components.

Definition 7. *Let* γ *and* δ *be two BCA-updates. We will say that update* γ dominates δ *(denoted as* $\gamma \geq \delta$*) if for any* $g_{uv} \in \mathbb{R}^{\mathcal{Y}^2}$ *it holds that* $\gamma[g_{uv}] \geq \delta[g_{uv}]$*, where the inequality is understood as component-wise inequalities* $\theta^{\gamma}_u[g_{uv}] \geq \theta^{\delta}_u[g_{uv}]$ *and* $\theta^{\gamma}_v[g_{uv}] \geq \theta^{\delta}_v[g_{uv}]$*.*

We can show the following dominance results. Recall that $\mathcal{U}$, $\mathcal{H}$ and $\mathcal{M}$ are the `uniform`, `MPLP` and `MPLP++` BCA-updates, respectively.

Proposition 2. *The following BCA-dominances hold:* $\mathcal{H} \geq \mathcal{M} \geq \mathcal{U}$*.*

It is easy to see that the dominance Definition 7 is transitive and so also $\mathcal{H} \geq \mathcal{U}$. We will prove that such coordinate-wise dominance of BCA-updates implies also the dominance in the dual objective whenever the following monotonicity property holds:

Definition 8. *A BCA-update* γ *is called* monotonous *if* $(\theta_u \geq \theta'_u,\ \theta_v \geq \theta'_v)$ *implies* $\gamma[\theta_u + \theta_{uv} + \theta_v] \geq \gamma[\theta'_u + \theta_{uv} + \theta'_v]$ *for all* θ, θ'*.*

Proposition 3. *Updates* $\mathcal{U}$ *and* $\mathcal{M}$ *are monotonous. The update* $\mathcal{H}$ *is not monotonous.*

With these results we can formulate our main claim about domination in the objective value for the whole iteration.

Theorem 2. *Let BCA-update γ dominate BCA-update μ and let μ be monotonous. Then a BCA-iteration with γ dominates a BCA-iteration with μ.*

From Proposition 2, Propostion 3 and Theorem 2 it follows now that BCA-iteration of `MPLP++` dominates that of `MPLP`, which in its turn dominates uniform.

6 Parallelization

To optimize $D(\theta^\phi)$, we have to perform local operations on a graph, that per reparametrization influence only one edge $uv \in \mathcal{E}$ and it's incident vertices u and v. The remaining graph $\mathcal{G}' = (\mathcal{V} - \{u, v\}, \mathcal{E} - \{I_u \cup I_v\})$, (where I_u and I_v are the index sets for vertices u and v) remains unchanged. This gives rise to opportunities for parallelization. However, special care has to be taken to prevent a *race condition* which occurs when two or more threads access shared data and they try to change it at the same time.

Consider the case of Fig. 3, choosing edges 1 and 2 or 1 and 6 to process in parallel. These edges have vertex A in common, which would lead to a race condition. Processing edges 1 and 3 in parallel would lead to more parallelization as there are no conflicting nodes.

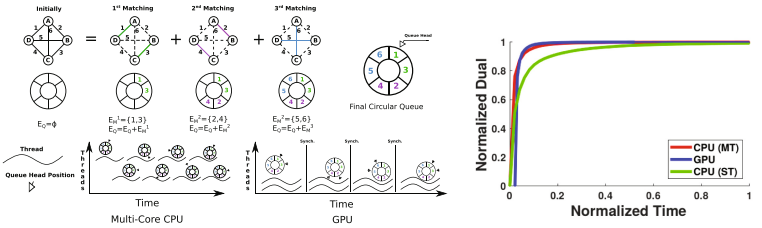
Finding such edges without intersecting vertices is a well-studied problem in combinatorial optimization [41]. A *matching* $\mathcal{M} \subset \mathcal{E}$ in graph $\mathcal{G}$ is a set of edges such that no two edges in $\mathcal{M}$ share a common vertex. A matching is *maximum* if it includes the largest number of edges, $|\mathcal{M}|$. Every edge in a matching can be processed in parallel without race conditions ensuing. There exist efficient greedy algorithms to find a maximum matching which we use. This gives rise to Algorithm 1 for covering all edges of the graph while ensuring good parallelization. To cover the entire graph, we call a matching algorithm repeatedly, until all edges are exhausted.

Initially, in line 1 the edge queue $\mathcal{Q}_\mathcal{E}$ is empty. In line 3, a maximum matching $\mathcal{E}_\mathcal{M}$ is found. This is added to $\mathcal{Q}_\mathcal{E}$ in line 4. This continues until all edges have been exhausted, *i.e.* the edges remaining $\mathcal{E}_\mathcal{R}$ is empty. The queue thus has a structure $\mathcal{Q}_\mathcal{E} = (\mathcal{E}^1_\mathcal{M}, \mathcal{E}^2_\mathcal{M}, ..., \mathcal{E}^n_\mathcal{M})$, ordered left to right. $\mathcal{E}^i_\mathcal{M}$ being the i^{th} matching computed. The threads running in parallel can keep popping edges from $\mathcal{Q}_\mathcal{E}$ and processing them without much need for mutex locking.

We have different implementation algorithms for GPUs and multi-core CPUs.

CPU Implementation: Modern CPUs consist of multiple cores with each core having one hardware thread. Hyper-threading allows for an additional lower-performing thread per core.

Processing an edge is a short-lived task and launching a separate thread for each edge would have excessive overhead. To process lightweight tasks we use the *thread-pool* design pattern. A thread-pool keeps multiple threads waiting for tasks to be executed concurrently by a supervising program. The threads are launched only once and continuously process tasks from a task-queue. As



(a) Parallelization Illustration (b) GPU vs CPU Performance

Fig. 3. This figure shows parallelization details and a performance comparison for CPU and GPU. (a) shows the details of how the edge schedule is computed for maximizing throughput. The first row shows how edge selection is carried out by finding matchings and adding the edges in these matchings to the queue. The second row shows how the threads are launched for the CPU and GPU. For the CPU, threads are launched dynamically at different time instances, and no synchronization is carried out across all threads. This is due to a local memory locking mechanism (mutex) for the CPU. For the GPU, many threads are launched simultaneously and synchronized simultaneously via a memory barrier. This barrier is shown as the vertical line with **Synch.** (b) shows the running time comparison between single-threaded, multi-threaded and GPU versions of the `MPLP++` algorithm. The GPU takes some time to load memory from the host (CPU) to device (GPU)..

Algorithm 1 Compute Edge Schedule

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Initial: $\mathcal{Q}_{\mathcal{E}} := \emptyset$ (Empty edge queue),
$\mathcal{E}_{\mathcal{R}} := \mathcal{E}$ (Initial pool of edges)
while $\mathcal{E}_{\mathcal{R}} ! = \emptyset$ **do**
  $\mathcal{E}_{\mathcal{M}} :=$ Maximum_Matching$(\mathcal{V}, \mathcal{E}_{\mathcal{R}})$
  $\mathcal{Q}_{\mathcal{E}}.push(\mathcal{E}_{\mathcal{M}})$ (Push maximum-matching $\mathcal{E}_{\mathcal{M}}$ to the queue)
  $\mathcal{E}_{\mathcal{R}} := \mathcal{E}_{\mathcal{R}} - \mathcal{E}_{\mathcal{M}}$ (Remove matched edges from $\mathcal{E}_{\mathcal{R}}$)
end while

they are launched only once, the latency in execution due to overhead in thread creation and destruction is avoided.

In the case of our algorithm the task queue is $\mathcal{Q}_{\mathcal{E}}$. The thread picks up the index of the edge to process and performs the `MPLP++` operation ($\mathcal{H}$). During the `MPLP++` operation the node and edge structures are locked by mutexes. One iteration of the algorithm is complete when all edges have been processed. Since multiple iterations may be required to reach convergence, our task queue is circular, letting the threads restart the reparameterization process from the first element of $\mathcal{Q}_{\mathcal{E}}$. The ordering of $\mathcal{Q}_{\mathcal{E}}$ prevents heavy lock contention of mutexes.

GPU Implementation: Unlike CPUs, GPUs do not use mutexes for synchronization. The threads in each GPU processor are synchronized via a hardware barrier synchronization. A barrier for a group of threads stops the threads at this point and prevents them from proceeding until all other threads/processes reach this barrier.

This is where the ordering of $\mathcal{Q}_{\mathcal{E}}$ comes in handy. Recall the structure of $\mathcal{Q}_{\mathcal{E}} = (\mathcal{E}^1_{\mathcal{M}}, \mathcal{E}^2_{\mathcal{M}}, ..., \mathcal{E}^n_{\mathcal{M}})$. Barrier synchronization can be used between the matchings $\mathcal{E}^i_{\mathcal{M}}$ and $\mathcal{E}^{i+1}_{\mathcal{M}}$, allowing for the completion of the `MPLP++` BCA update operations for $\mathcal{E}^i_{\mathcal{M}}$, before beginning the processing of $\mathcal{E}^{i+1}_{\mathcal{M}}$. This minimizes the time threads spend waiting while the other threads complete, as they have no overlapping areas to write to in the memory.

7 Experimental Evaluation

In our experiments, we use a 4-core Intel i7-4790K CPU @ 4.00GHz, with hyper-threading, giving 8 logical cores. For GPU experiments, we used the NVIDIA Tesla K80 GPU with 4992 cores.

Compared Algorithms. We compare different algorithms in two regimes: the wall-clock time and the machine-independent regime, where we count the number of operations performed. For the latter one we use the notion of the *oracle call*, which is an operation like $\min_{t \in \mathcal{Y}} g_{uv}(s,t),\ \forall s \in \mathcal{Y}$ involving a single evaluation of all costs of a pairwise factor. When speaking about *oracle complexity* we mean the number of oracle calls per single iteration of the algorithm. As different algorithms have different oracle complexities we define a *normalized iteration* as exactly $|\mathcal{E}|$ messages for an even comparison across algorithms. We compare the following `BCA` schemes:

- The Tree-Reweighted Sequential (`TRWS`) message passing algorithm [17] has consistently been the best performing method on several benchmarks like [14]. Its oracle complexity is $2|\mathcal{E}|$. We use the multicore implementation introduced in [42] for comparison, which is denoted as `TRWS(MT)` when run on multiple cores.
- The Max-Product Linear Programming (`MPLP`) algorithm [7] with BCA-updates ($\mathcal{M}$). It's oracle complexity is thus $2|\mathcal{E}|$. For `MPLP` we have our own multi-threaded implementation that is faster than the original one by a factor of 4.
- The Min-Sum Diffusion Algorithm (`MSD`) [36], is one of the earliest (in the 70s) proposed `BCA` algorithms for graphical models. The oracle complexity of the method is $4|\mathcal{E}|$.
- The `MPLP++` algorithm with BCA-updates ($\mathcal{H}$) has the oracle complexity of $3|\mathcal{E}|$. The algorithm is parallelized as described in Sect. 6 for both CPU and GPU. The corresponding legends are `MPLP++(MT)` and `MPLP++(GPU)`.

Dense Datasets. To show the strength of our method we consider the following datasets with underlying densely connected graphs:

- The `worms` dataset [11] consists of 30 problem instances coming from the field of bioimaging. The problems' graphs are dense but not fully connected with about $0.1|\mathcal{V}|^2$ of edges, up to 600 nodes and up to 1500 labels.
- The `protein`-folding dataset [48] taken from the OpenGM benchmark [14] has 21 problem instances with 33 to 1972 variables. The models are fully connected and have 81 to 503 labels per node.

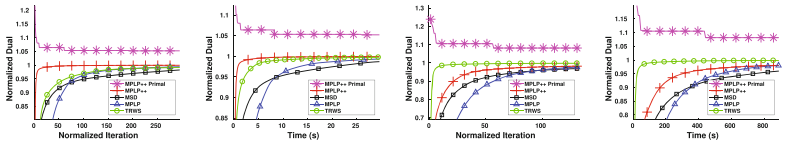
– `pose` is the dataset inspired by the recent work [29] showing state-of-the-art performance on the 6D object pose estimation problem. In [29] this problem is formulated as MAP-inference in a fully connected graphical model. The set of nodes of the underlying graph coincides with the set of pixels of the input image (up to some downscale factor), which requires specialized heuristics to solve the problem. Contrary to the original work, we assume that the position of the object is given by its mask (we used ground-truth data from the validation set, but assume the mask could be provided by some segmentation method) and treat only the pixels inside the mask as the nodes of the corresponding graphical model. The unary and pairwise costs are constructed in the same way as in [29] but with different hyper-parameters. This dataset has 32 problem instances with 600 to 4800 variables each and 13 labels per node. The models are all fully connected.

Sparse Datasets. Although our method is not the best performing one on sparse graphical models, we include the comparison on the following four-connected grid-graph based benchmark datasets for fairness:

– `color-seg` from [14] has Potts pairwise costs. The nodes contain up to 12 labels.
– `stereo` from the Middlebury MRF benchmark [45] consists of 3 models with truncated linear pairwise costs and 16, 20 and 60 labels respectively.

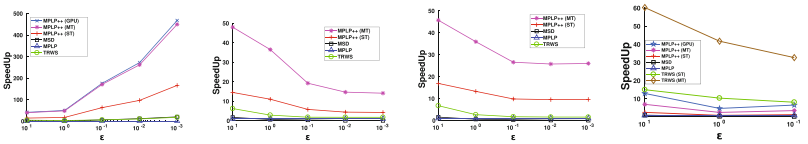
Algorithm Convergence. Figure 4 shows convergence of the considered algorithms in a sequential setting for the `protein` and `stereo` datasets as representatives of the dense and sparse problems. For other datasets the behavior is similar, therefore we moved the corresponding plots to the supplement. The proposed `MPLP++` method outperforms all its competitors on *all* dense problem instances and is inferior to `TRWS` only on sparse ones. This holds for both comparisons: the implementation-independent *normalized iteration* and the implementation-dependent *running time* ones. Figure 5 shows relative time speed-ups of the considered methods as a function of the attained dual precision. The speed of `MPLP`, as typically the slowest one, is taken to be 1, *i.e.*, for other algorithms the speed-up compared to `MPLP` is plotted. This includes also the CPU-parallel versions of `MPLP++`, `TRWS` and the GPU-parallel `MPLP++`. Figure 5 also shows that for dense models `MPLP++` is 2 to 10 times faster than `TRWS` in the sequential setting and 7 to 40 times in the parallel setting. The speed-up w.r.t. `MPLP` is 5 to 150 times depending on the dataset and the required precision.

Performance Degradation with Graph Sparsification. In this test we gradually and randomly removed edges from the graphs of the `pose` dataset and measured performance of all algorithms. Figure 6 shows that down to atleast 10% of all possible edges `MPLP++` leads the board. Only when the number of edges drops to 5% and less, `TRWS` starts to outperform `MPLP++`. Note, the density of edges in grid graphs considered above does not exceed the 0.007% level.



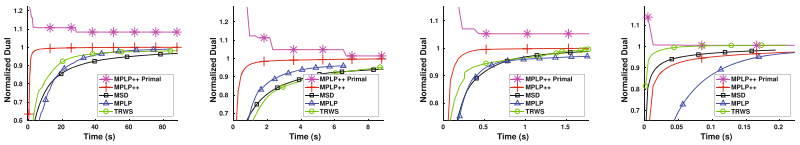
(a) protein, iter (b) protein, sec (c) stereo, iter (d) stereo, sec

Fig. 4. Improvement in dual as a function of time and iterations for the protein-folding and stereo dataset. The algorithms we compare have different message-passing schemes and end up doing different amounts of work per iteration. Hence, for a fair comparison across algorithms we define a normalized iteration as exactly $|\mathcal{E}|$ messages. This is also equal to number of messages passed in a normal iteration divided by its oracle complexity. (a) and (b) are for the dense protein-folding dataset, where both per unit time and per iteration, MPLP++ outperforms TRWS and all other algorithms. In the sparse stereo dataset (c,d) TRWS beats all other algorithms. Results are averaged over the entire dataset. The dual is normalized to 1 for equal weighing of every instance in the dataset.



(a) pose (b) protein (c) worms (d) stereo

Fig. 5. Relative speed w.r.t the MPLP algorithm in converging to within ε of the best attained dual optima $D^*(\theta^\phi)$, i.e. $D^*(\theta^\phi) - \varepsilon$. The plot shows the speedups of all the algorithms relative to MPLP for ε's 0.001, 0.01, 0.1, 1 and 10 of $D^*(\theta^\phi)$. Figure (a) shows MPLP++ is 50× faster than MPLP in converging to within 10% of $D^*(\theta^\phi)$. Figure (b) and (c) likewise show an order of magnitude speedup. Figure (d) shows that for the stereo dataset consisting of sparse graphs TRWS dominates MPLP++. Convergence only till 0.1% of $D^*(\theta^\phi)$ is shown for stereo as only TRWS converges to the required precision.



(a) 100% (b) 40% (c) 10% (d) 5%

Fig. 6. Degradation with Sparsity (Dual vs Time): (a)–(d) show graphs with decreasing average connectivity given as percentage of possible edges in the figure subcaption. In (a)–(c) MPLP++ outperforms TRWS. MPLP++ is resilient to graph sparsification even when 90% of the edges have been removed. Only when more than 95% of the edges have been removed as in (d) TRWS outperforms MPLP++.

8 Conclusions and Outlook

Block-coordinate ascent methods remain a perspective research direction for creating efficient parallelizable dual solvers for MAP-inference in graphical models. We have presented one such solver beating the state-of-the-art on dense graphical models with arbitrary potentials. The method is directly generalizable to higher order models, which we plan to investigate in the future.

Acknowledgements. A. Shekhovtsov was supported by Czech Science Foundation grant 18-25383S.

References

1. Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. Algorithmica **2**(1), 195–208 (1987). https://doi.org/10.1007/BF01840359
2. Bergtholdt, M., Kappes, J., Schmidt, S., Schnörr, C.: A study of parts-based object class detection using complete graphs. Int. J. Comput. Vis. **87**(1–2), 93 (2010)
3. Chen, L.C., Schwing, A., Yuille, A., Urtasun, R.: Learning deep structured models. In: International Conference on Machine Learning, pp. 1785–1794 (2015)
4. Chen, W., Wada, K.: On computing the upper envelope of segments in parallel. In: Proceedings. 1998 International Conference on Parallel Processing (Cat. No. 98EX205), pp. 253–260, August 1998. https://doi.org/10.1109/ICPP.1998.708493
5. Choi, J., Rutenbar, R.A.: Hardware implementation of MRF map inference on an FPGA platform. In: 2012 22nd International Conference on Field Programmable Logic and Applications (FPL), pp. 209–216. IEEE (2012)
6. Cooper, M., Schiex, T.: Arc consistency for soft constraints. Artif. Intell. **154**(1–2), 199–227 (2004)
7. Globerson, A., Jaakkola, T.S.: Fixing max-product: convergent message passing algorithms for MAP LP-relaxations. In: Advances in Neural Information Processing Systems, vol. 20 (2008)
8. Hazan, T., Shashua, A.: Norm-Product Belief Propagation: Primal-Dual Message-Passing for approximate inference (2008)
9. Hurkat, S., Choi, J., Nurvitadhi, E., Martínez, J.F., Rutenbar, R.A.: Fast hierarchical implementation of sequential tree-reweighted belief propagation for probabilistic inference. In: 2015 25th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–8. IEEE (2015)
10. Hurley, B., et al.: Multi-language evaluation of exact solvers in graphical model discrete optimization. Constraints **21**(3), 413–434 (2016)
11. Kainmueller, D., Jug, F., Rother, C., Meyers, G.: Graph matching problems for annotating c. elegans (2017). https://doi.org/10.15479/AT:ISTA:57. Accessed 10 Sept 2017
12. Kainmueller, D., Jug, F., Rother, C., Myers, G.: Active graph matching for automatic joint segmentation and annotation of *C. elegans*. In: Golland, P., Hata, N., Barillot, C., Hornegger, J., Howe, R. (eds.) MICCAI 2014. LNCS, vol. 8673, pp. 81–88. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10404-1_11
13. Kannan, H., Komodakis, N., Paragios, N.: Newton-type methods for inference in higher-order markov random fields. In: IEEE International Conference on Computer Vision and Pattern Recognition (2017)

14. Kappes, J.H., et al.: A comparative study of modern inference techniques for structured discrete energy minimization problems. Int. J. Comput. Vis., 1–30 (2015). https://doi.org/10.1007/s11263-015-0809-x
15. Kappes, J.H., Savchynskyy, B., Schnörr, C.: A bundle approach to efficient MAP-inference by lagrangian relaxation. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1688–1695. IEEE (2012)
16. Kirillov, A., Schlesinger, D., Zheng, S., Savchynskyy, B., Torr, P.H.S., Rother, C.: Joint training of generic CNN-CRF models with stochastic optimization. In: Lai, S.-H., Lepetit, V., Nishino, K., Sato, Y. (eds.) ACCV 2016. LNCS, vol. 10112, pp. 221–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54184-6_14
17. Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. IEEE Trans. Pattern Anal. Mach. Intell. **28**(10), 1568–1583 (2006)
18. Kolmogorov, V.: A new look at reweighted message passing. IEEE Trans. Pattern Anal. Mach. Intell. **37**(5), 919–930 (2015)
19. Kolmogorov, V., Rother, C.: Comparison of energy minimization algorithms for highly connected graphs. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3952, pp. 1–15. Springer, Heidelberg (2006). https://doi.org/10.1007/11744047_1
20. Kolmogorov, V., Thapper, J., Zivny, S.: The power of linear programming for general-valued CSPs. SIAM J. Comput. **44**(1), 1–36 (2015)
21. Komodakis, N., Paragios, N., Tziritas, G.: MRF optimization via dual decomposition: message-passing revisited. In: IEEE 11th International Conference on Computer Vision, ICCV 2007, pp. 1–8. IEEE (2007)
22. Kovalevsky, V., Koval, V.: A diffusion algorithm for decreasing energy of max-sum labeling problem. Glushkov Institute of Cybernetics, Kiev, USSR (1975). unpublished
23. Krähenbühl, P., Koltun, V.: Efficient inference in fully connected CRFs with gaussian edge potentials. In: Advances in Neural Information Processing Systems, pp. 109–117 (2011)
24. Li, M., Shekhovtsov, A., Huber, D.: Complexity of discrete energy minimization problems. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9906, pp. 834–852. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46475-6_51
25. Luong, D.V.N., Parpas, P., Rueckert, D., Rustem, B.: Solving MRF minimization by mirror descent. In: Bebis, G., et al. (eds.) ISVC 2012. LNCS, vol. 7431, pp. 587–598. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33179-4_56
26. Martins, A.F.T., Figueiredo, M.A.T., Aguiar, P.M.Q., Smith, N.A., Xing, E.P.: An augmented lagrangian approach to constrained MAP inference. In: ICML (2011)
27. Meshi, O., Globerson, A.: An alternating direction method for dual MAP LP relaxation. In: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (eds.) ECML PKDD 2011. LNCS (LNAI), vol. 6912, pp. 470–483. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23783-6_30
28. Meshi, O., Globerson, A., Jaakkola, T.S.: Convergence rate analysis of MAP coordinate minimization algorithms. In: Advances in Neural Information Processing Systems, pp. 3014–3022 (2012)
29. Michel, F., et al.: Global hypothesis generation for 6D object pose estimation. arXiv preprint (2017)
30. Nowozin, S., Rother, C., Bagon, S., Sharp, T., Yao, B., Kohli, P.: Decision Tree Fields. In: 2011 IEEE International Conference on Computer Vision (ICCV), pp. 1668–1675. IEEE (2011)

31. Průša, D., Werner, T.: LP relaxation of the potts labeling problem is as hard as any linear program. IEEE Trans. Pattern Anal. Mach. Intell. **39**(7), 1469–1475 (2017)
32. Prusa, D., Werner, T.: Universality of the local marginal polytope. PAMI **37**(4), April 2015
33. Ravikumar, P., Agarwal, A., Wainwright, M.: Message-passing for graph-structured linear programs: proximal methods and rounding schemes. JMLR **11**, 1043–1080 (2010)
34. Savchynskyy, B., Kappes, J., Schmidt, S., Schnörr, C.: A study of Nesterov's scheme for Lagrangian decomposition and MAP labeling. In: 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1817–1823. IEEE (2011)
35. Savchynskyy, B., Schmidt, S., Kappes, J., Schnörr, C.: Efficient MRF energy minimization via adaptive diminishing smoothing. arXiv preprint arXiv:1210.4906 (2012)
36. Schlesinger, M., Antoniuk, K.: Diffusion algorithms and structural recognition optimization problems. Cybern. Syst. Anal. **47**(2), 175–192 (2011)
37. Schlesinger, M., Giginyak, V.: Solution to structural recognition (max,+)-problems by their equivalent transformations. in 2 Parts. Control Syst, Comput. (1–2) (2007)
38. Schlesinger, M.I.: Syntactic analysis of two-dimensional visual signals in noisy conditions. Kibernetika **4**(113–130), 1 (1976)
39. Schlesinger, M.I., Flach, B.: Some solvable subclasses of structural recognition problems. In: Czech Pattern Recognition Workshop. 2000, pp. 55–62 (2000)
40. Schmidt, S., Savchynskyy, B., Kappes, J., Schnörr, C.: Evaluation of a first-order primal-dual algorithm for MRF energy minimization. In: EMMCVPR 2011 (2011)
41. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency, vol. 24. Springer, Heidelberg (2003)
42. Shekhovtsov, A., Swoboda, P., Savchynskyy, B.: Maximum persistency via iterative relaxed inference with graphical models. PAMI (2017)
43. Shekhovtsov, A., Reinbacher, C., Graber, G., Pock, T.: Solving dense image matching in real-time using discrete-continuous optimization. In: CVWW, p. 13 (2016)
44. Storvik, G., Dahl, G.: Lagrangian-based methods for finding MAP solutions for MRF models. IEEE Trans. Image Process. **9**(3), 469–479 (2000)
45. Szeliski, R., et al.: A comparative study of energy minimization methods for markov random fields with smoothness-based priors. IEEE Trans. Pattern Anal. Mach. Intell. **30**(6), 1068–1080 (2008)
46. Thapper, J., Živný, S.: The power of linear programming for valued CSPs. In: Symposium on Foundations of Computer Science (FOCS), pp. 669–678 (2012)
47. Werner, T.: A linear programming approach to max-sum problem: A review. IEEE Trans. Pattern Anal. Mach. Intell. **29**(7) (2007)
48. Yanover, C., Schueler-Furman, O., Weiss, Y.: Minimizing and learning energy functions for side-chain prediction. J. Comput. Biol. **15**(7), 899–911 (2008)
49. Živný, S., Werner, T., Průša, D.a.: The Power of LP Relaxation for MAP Inference, pp. 19–42. The MIT Press, Cambridge (2014)