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

Siddharth Tourani[1], Alexander Shekhovtsov[2], Carsten Rother [1], Bogdan Savchynskyy[1]

[1] Visual Learning Lab, IWR, Uni. Heidelberg, Germany
[2] Centre for Machine Perception,Czech Technical University,Prague, Czech Republic

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 the low-performing solver, the Max Product Linear Programming (MPLP) algorithm [1], 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 [2]. 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, MAP-Inference, Block-Coordinate-Ascent, Dense Graphs, 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 [3], significantly improving the image segmentation field, once an efficient solver for the model was proposed. More recently, 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 robustness potentials. A prominent application of the non-Gaussian fully-connected CRF case achieved state-of-the-art performance in 6D object pose estimation problem [4], with an efficient *application-specific* solver. Other examples of densely connected models were proposed in the area of stereo-reconstruction [5], body pose estimation [6, 7, 8], bio-informatics [9] 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 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 [10] 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) [11]. Although there are multiple algorithms addressing the MAP inference, which we discuss in § 2, the linear programs obtained by relaxing the MAP-inference problem are not any simpler than general linear programs [12]. 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 [2] is probably the most well-known and efficient solver of this class, as shown in [13]. 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 [1] (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 [14] 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 [15] 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 [2], which is exploited in specialized implementations [16, 17] 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 § 4 and [18]).

The new MPLP++ method consistently outperforms all its competitors, including TRW-S [2], 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 [4]. Our code and the new benchmark dataset will be made publicly available together with the paper.

2 Related work

The general MAP inference problem for discrete graphical models (formally defined in § 3) is NP-hard and is also hard to approximate [19]. A natural linear programming relaxation is obtained by formulating it as a 0-1 integer linear program (ILP) and relaxing the integrality constraints [20] (see also the recent review [21]). A hierarchy of relaxations is known [22], from which the so-called Base LP relaxation, also considered in our work, is the simplest one. It was shown [11, 23] that this relaxation is tight for all tractable subclasses of the problem. For many other classes of problems it provides approximation guarantees, reviewed [19]. A large number of specialized algorithms were developed for this relaxation.

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 [24, 25, 26, 27], dual sub-gradient [28, 29, 30], bundle [31], mirror-descent [32] smoothing-based [33, 34, 35] and (quasi-) Newton [36] methods.

However, as it was shown in [37], the linear programs arising from the relaxation of the MAP inference have the same computational complexity as general LPs. At the same time, if the problem is not known to belong to a tractable class, solving the relaxation to optimality may be of low practical utility. A comparative study [13] notes that TRW-S [2] 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 [1], min-sum diffusion [38, 15] and DualMM [14] 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. Suboptimal solutions of the relaxation can also be employed in exact solvers [40, 41] using cutting plane or branch-and-cut techniques and in methods identifying a part of optimal solution [42, 43].

3 Preliminaries

Notation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denotes an undirected graph, with vertex set $\mathcal{V}$ (we assume $\mathcal{V} = \{1, \dots |\mathcal{V}|\}$) 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 a triple $(\mathcal{G}, \mathcal{Y}^{\mathcal{V}}, \theta)$ consists in finding the labelling with the smallest total cost, *i.e.*:

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

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 [19]. A number of approaches to tackle it in different practical scenarios is reviewed in [13, 44]. One of the widely applicable techniques is based on (approximately) solving its linear programming (LP) relaxation as discussed in § 2.

Dual Problem Most existing solvers for the LP relaxation tackle its dual form, which we introduce now. This is because the LP dual has much fewer variables than the primal and can also be written in the form of unconstrained concave (piecewise-linear) maximization. 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. [21]) 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* [45] or *equivalent transformations* [20].

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 (*e.g.* $E(y|\theta) = E(y|\theta^\phi)$), the lower bound $D(\theta^\phi)$ depends on reparametrization, which is $D(\theta) \neq D(\theta^\phi)$. Therefore, a natural maximization problem arises as maximization of the lower bound over all equivalent costs: $\max_\phi D(\theta^\phi)$. It is known (e.g. [21]) 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, piece-wise 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 § 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 [2], [14] or their generalizations [46].

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 [15], as well as in *the convex message passing* [47], and [46] 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 [1]. 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:

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

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 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 [21] or weak tree-agreement [2] 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.

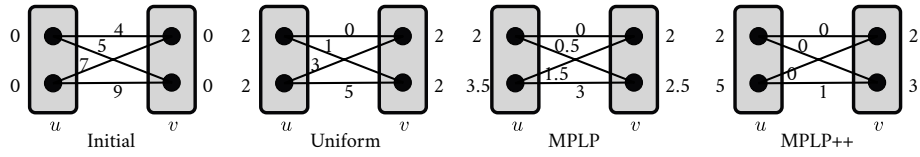


Fig. 1. Illustration of the considered BCA-updates. The gray boxes in the figure represent graph nodes. The black dots in them the labels. The edges connecting the black dots make up the pairwise costs. The numbers adjacent to the edges and labels are the pairwise and unary costs respectively.

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 on 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 $\bar{\phi} := \bar{\phi} + \phi$.

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 be then 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 $\theta^{\gamma}_v = \phi_{u\to v}$.*

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 $\mathcal{U}$:

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

[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.

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 $\theta_u^{\mathcal{U}}(\cdot)$ and $\theta_v^{\mathcal{U}}(\cdot)$ are constant and therefore any pairwise minimizer of $\theta_{uv}^{\mathcal{U}}$ is consistent with them.

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

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

The MPLP algorithm [1] 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 [14]. It is given by the following procedure defining the reparametrization mapping $\mathcal{H}$:

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

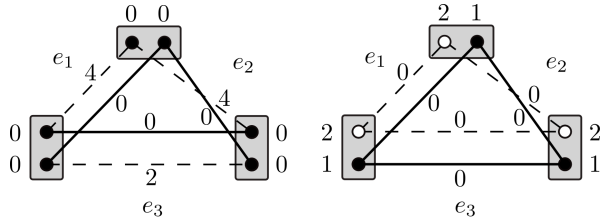
In other words, the MPLP++ update first performs the MPLP update and then pushes as much cost from the pairwise factor to the unary ones, as needed to fulfill $\min_t \theta_{uv}^{\mathcal{M}}(s,t) = \min_s \theta_{uv}^{\mathcal{M}}(s,t) = 0$ for all labels s and t in the nodes u and v respectively. 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. Fig. 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 example in Fig. 2. Therefore, proving only that some algorithm is a BCA, does not imply its efficiency.

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.*, [48]. Using $|\mathcal{Y}|$ processors, the computation time can be further reduced to $O(\log|\mathcal{Y}|)$ with appropriate parallel algorithms, *e.g.*, [18].

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 [2]. Assuming we have already computed a primal integer solution x_v^* for all $v < u$, we want to compute x_u^*. To do so, we use the following equation for the assignment

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



(a) Uniform (b) MPLP

Fig. 2. Choosing the right BCA-update is important. Notation has the same meaning as in Fig. 1, black circles denote locally optimal labels, white circles - the non-optimal ones. 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.

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 the *arc-consistency* [21] and the *weak tree-agreement* [2].

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 the node-edge agreement, which is a special case of the weak tree agreement [2] when individual nodes and edges are considered as the trees in a problem decomposition. This condition is also known as a non-empty *kernel* [21] / *arc-consistent closure* [45] of the cost vector θ.

Definition 4. *We will say that the costs $\theta \in \mathbb{R}^{\mathcal{I}}$ fulfill the* 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.*, primal is feasible). However, such convergence is weaker than convergence in the reparametrization than is desired. 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 [15].

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

Another important question is the comparison of different BCA methods. When comparing different algorithms, an ultimate goal is to prove faster convergence of one compared to the other one. 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, considered algorithms are all of the same family and it is likely that their asymptotic rates are the same. Instead, we study the *dominance*, the 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 § 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, Proposition 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 $\mathcal{G}' = (\mathcal{V} - \{u, v\}, \mathcal{E} - \{I_u \cup I_v\})$ remains unchanged. This gives rise to opportunities for parallelization. However, special care has to be taken to prevent *race conditions* 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 race conditions. Processing edges 1 and 3 in parallel would lead to more parallelization as there are no conflicting nodes. Thus, for maximal parallelization we have to come up with an ordering of edges where threads working in parallel do not have common vertices.

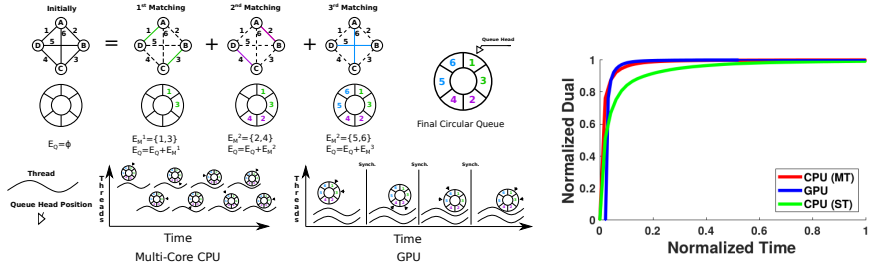
Finding such edges without intersecting vertices is a well-studied problem in combinatorial optimization [49]. A *matching* $\mathcal{M} \subset \mathcal{E}$ in graph $\mathcal{G}$ is a set of edges such that no two edges in $\mathcal{M}$ share a common vertex. A matching is *maximum* if it includes 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 threads. Hyper-threading allows for an additional thread per core, but with lesser performance in the second thread compared to the first.

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



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

Fig. 3. Figure shows parallelization details and 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). This is why it takes longer to get started than the CPU versions.

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

continuously process tasks from a task-queue. As 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 maybe 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 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 hyperthreading, 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 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 processing [2] algorithm has consistently been the best performing method on several benchmarks like [13]. 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 [1] algorithm 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 [15], 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 § 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:

- worms dataset [50] consists of 30 problem instances coming from the field of bioimaging. The problems' graphs are dense but not fully connected with about $0.1 \cdot \mathcal{V}^2$ of edges, up to 600 nodes and up to 1500 labels.
- protein-folding dataset [51] taken from the OpenGM benchmark [13]. This dataset has 21 problem instances with $33 - 1972$ variables. The models are fully connected and have $81 - 503$ labels per node.
- pose is the dataset inspired by the recent work [4] showing state-of-the-art performance on the 6D object pose estimation problem. In [4] 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 obtain practically useful solutions in reasonable time due to the large problem size. 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. Otherwise, the unary and pairwise costs are constructed in the same way as in [4]. We have got the learned unary costs from the authors of [4] and merely tuned the hyper-parameters on the validation set. This dataset has 32 problem instances with $600 - 4800$ 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:

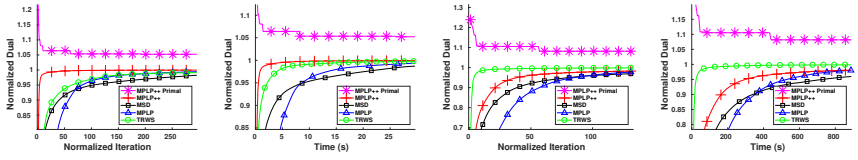
- `color-seg` from [13] has Potts pairwise costs. The nodes contain up to 12 labels.
- `stereo` from the Middlebury MRF benchmark [52]. This dataset consists of 3 models with truncated linear pairwise costs and 16, 20 and 60 labels respectively.

Algorithm Convergence Fig. 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 § A. 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. Fig. 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++`. Fig. 5 also shows that for dense models `MPLP++` is $2 - 10$ faster than `TRWS` in the sequential setting and $7 - 40$ times in the parallel setting. The speed-up w.r.t. `MPLP` is $5 - 150$ 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 measures performance of all algorithms. Fig. 6 shows that up to at least 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.

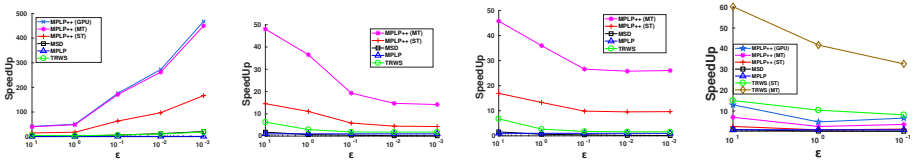
8 Conclusions and Outlook

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.



(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.

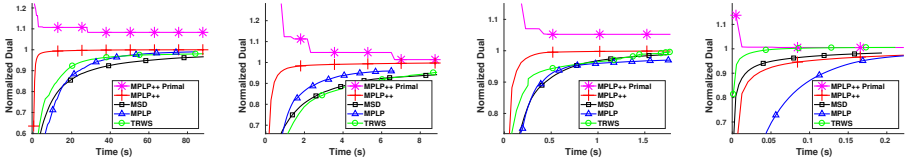


(a) pose (b) protein (c) worms (d) stereo

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 $50\times$ 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.

9 Acknowledgements

This project has received funding from the ERC under the European Unions Horizon 2020 research and innovation program (grant agreement No 647769). A. Shekhovtsov was supported by Czech Science Foundation grant 18-25383S.



(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 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++`.

References

[1] Globerson, A., Jaakkola, T.S.: Fixing Max-Product: Convergent Message Passing Algorithms for MAP LP-Relaxations. In: Advances in Neural Information Processing Systems 20. (2008)

[2] Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. IEEE transactions on pattern analysis and machine intelligence **28**(10) (2006) 1568–1583

[3] Krähenbühl, P., Koltun, V.: Efficient inference in fully connected CRFs with gaussian edge potentials. In: Advances in neural information processing systems. (2011) 109–117

[4] Michel, F., Kirillov, A., Brachmann, E., Krull, A., Gumhold, S., Savchynskyy, B., Rother, C.: Global hypothesis generation for 6D object pose estimation. arXiv preprint (2017)

[5] Kolmogorov, V., Rother, C.: Comparison of energy minimization algorithms for highly connected graphs. In: European Conference on Computer Vision, Springer (2006) 1–15

[6] Bergtholdt, M., Kappes, J., Schmidt, S., Schnörr, C.: A study of parts-based object class detection using complete graphs. International journal of computer vision **87**(1-2) (2010) 93

[7] Nowozin, S., Rother, C., Bagon, S., Sharp, T., Yao, B., Kohli, P.: Decision Tree Fields. In: Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE (2011) 1668–1675

[8] Kirillov, A., Schlesinger, D., Zheng, S., Savchynskyy, B., Torr, P.H., Rother, C.: Joint training of generic CNN-CRF models with stochastic optimization. In: Asian Conference on Computer Vision, Springer (2016) 221–236

[9] Kainmueller, D., Jug, F., Rother, C., Myers, G.: Active graph matching for automatic joint segmentation and annotation of C. elegans. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer (2014) 81–88

[10] Chen, L.C., Schwing, A., Yuille, A., Urtasun, R.: Learning deep structured models. In: International Conference on Machine Learning. (2015) 1785–1794

[11] Kolmogorov, V., Thapper, J., Zivny, S.: The power of linear programming for general-valued CSPs. SIAM Journal on Computing **44**(1) (2015) 1–36

[12] 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) (2017) 1469–1475

[13] Kappes, J.H., Andres, B., Hamprecht, F.A., Schnörr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B.X., Kröger, T., Lellmann, J., Komodakis, N., Savchynskyy, B., Rother, C.: A comparative study of modern inference techniques for structured discrete energy minimization problems. International Journal of Computer Vision (2015) 1–30

[14] Shekhovtsov, A., Reinbacher, C., Graber, G., Pock, T.: Solving dense image matching in real-time using discrete-continuous optimization. In: CVWW. (2016) 13

[15] Schlesinger, M., Antoniuk, K.: Diffusion algorithms and structural recognition optimization problems. Cybernetics and Systems Analysis **47**(2) (2011) 175–192

[16] Choi, J., Rutenbar, R.A.: Hardware implementation of mrf map inference on an fpga platform. In: Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on, IEEE (2012) 209–216

[17] 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: Field Programmable Logic and Applications (FPL), 2015 25th International Conference on, IEEE (2015) 1–8

[18] Chen, W., Wada, K.: On computing the upper envelope of segments in parallel. In: Proceedings. 1998 International Conference on Parallel Processing (Cat. No.98EX205). (Aug 1998) 253–260

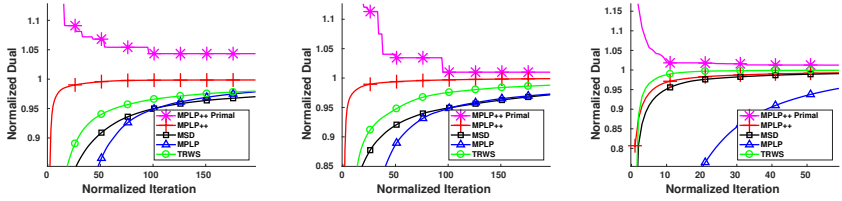
[19] Li, M., Shekhovtsov, A., Huber, D.: Complexity of discrete energy minimization problems. In: European Conference on Computer Vision. (2016) 834–852

[20] Schlesinger, M.I.: Syntactic analysis of two-dimensional visual signals in noisy conditions. Kibernetika **4**(113-130) (1976) 1
[21] Werner, T.: A linear programming approach to max-sum problem: A review. IEEE transactions on pattern analysis and machine intelligence **29**(7) (2007)
[22] Živný, S., Werner, T., Průša, D.a. In: The Power of LP Relaxation for MAP Inference. The MIT Press, Cambridge, USA (December 2014) 19–42
[23] Thapper, J., Živný, S.: The power of linear programming for valued CSPs. In: Symposium on Foundations of Computer Science (FOCS). (2012) 669–678
[24] Ravikumar, P., Agarwal, A., Wainwright, M.: Message-passing for Graph-structured Linear Programs: Proximal Methods and Rounding Schemes. JMLR **11** (2010) 1043–1080
[25] 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)
[26] Meshi, O., Globerson, A.: An Alternating Direction method for Dual MAP LP Relaxation. In: ECML/PKDD (2). (2011) 470–483
[27] Schmidt, S., Savchynskyy, B., Kappes, J., Schnörr, C.: Evaluation of a first-order primal-dual algorithm for mrf energy minimization. In: EMMCVPR 2011. (2011)
[28] Storvik, G., Dahl, G.: Lagrangian-based methods for finding MAP solutions for MRF models. IEEE Transactions on Image Processing **9**(3) (2000) 469–479
[29] Schlesinger, M., Giginyak, V.: Solution to structural recognition (max,+)-problems by their equivalent transformations. in 2 Parts. Control Systems and Computers (1-2) (2007)
[30] Komodakis, N., Paragios, N., Tziritas, G.: MRF optimization via dual decomposition: Message-passing revisited. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, IEEE (2007) 1–8
[31] Kappes, J.H., Savchynskyy, B., Schnörr, C.: A bundle approach to efficient MAP-inference by lagrangian relaxation. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE (2012) 1688–1695
[32] Luong, D.V., Parpas, P., Rueckert, D., Rustem, B.: Solving MRF minimization by mirror descent. In: International Symposium on Visual Computing, Springer (2012) 587–598
[33] Savchynskyy, B., Kappes, J., Schmidt, S., Schnörr, C.: A study of Nesterov's scheme for Lagrangian decomposition and MAP labeling. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE (2011) 1817–1823
[34] Savchynskyy, B., Schmidt, S., Kappes, J., Schnörr, C.: Efficient MRF energy minimization via adaptive diminishing smoothing. arXiv preprint arXiv:1210.4906 (2012)
[35] Meshi, O., Globerson, A., Jaakkola, T.S.: Convergence rate analysis of MAP coordinate minimization algorithms. In: Advances in Neural Information Processing Systems. (2012) 3014–3022
[36] 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)
[37] Prusa, D., Werner, T.: Universality of the local marginal polytope. PAMI **37**(4) (April 2015)
[38] Kovalevsky, V., Koval, V.: A diffusion algorithm for decreasing energy of max-sum labeling problem. Glushkov Institute of Cybernetics, Kiev, USSR (1975) Unpublished.
[39] Schlesinger, M.I., Flach, B.: Some solvable subclasses of structural recognition problems. In: Czech Pattern Recognition Workshop. Volume 2000. (2000) 55–62
[40] Cooper, M.C., De Givry, S., Sánchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence **174**(7-8) (2010) 449–478
[41] Savchynskyy, B., Kappes, J.H., Swoboda, P., Schnörr, C.: Global MAP-optimality by shrinking the combinatorial search area with convex relaxation. In: Advances in Neural Information Processing Systems. (2013) 1950–1958
[42] Shekhovtsov, A., Swoboda, P., Savchynskyy, B.: Maximum persistency via iterative relaxed inference with graphical models. PAMI (2017)

[43] Swoboda, P., Shekhovtsov, A., Kappes, J.H., Schnorr, C., Savchynskyy, B.: Partial optimality by pruning for MAP-inference with general graphical models. PAMI **38**(7) (2016) 1370–1382
[44] Hurley, B., OSullivan, B., Allouche, D., Katsirelos, G., Schiex, T., Zytnicki, M., De Givry, S.: Multi-language evaluation of exact solvers in graphical model discrete optimization. Constraints **21**(3) (2016) 413–434
[45] Cooper, M., Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence **154**(1-2) (2004) 199–227
[46] Kolmogorov, V.: A new look at reweighted message passing. IEEE transactions on pattern analysis and machine intelligence **37**(5) (2015) 919–930
[47] Hazan, T., Shashua, A.: Norm-Product Belief Propagation: Primal-Dual Message-Passing for approximate inference (2008)
[48] Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. Algorithmica **2**(1) (Nov 1987) 195–208
[49] Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Volume 24. Springer Science & Business Media (2003)
[50] Kainmueller, D., Jug, F., Rother, C., Meyers, G.: Graph matching problems for annotating c. elegans. http://dx.doi.org/10.15479/AT:ISTA:57 (2017) Accessed: 2017-09-10.
[51] Yanover, C., Schueler-Furman, O., Weiss, Y.: Minimizing and learning energy functions for side-chain prediction. Journal of Computational Biology **15**(7) (2008) 899–911
[52] Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., Rother, C.: A comparative study of energy minimization methods for markov random fields with smoothness-based priors. IEEE transactions on pattern analysis and machine intelligence **30**(6) (2008) 1068–1080
[53] Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
[54] Bartle, R., Sherbert, D.: Introduction to real analysis. John Wiley & Sons Canada, Limited (2000)

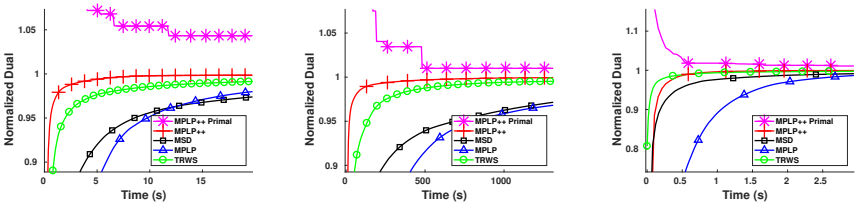
MPLP++: Fast, Parallel Dual Block-Coordinate Ascent for Dense Graphical Models (ECCV'18 Appendix)

A Additional Experimental Results



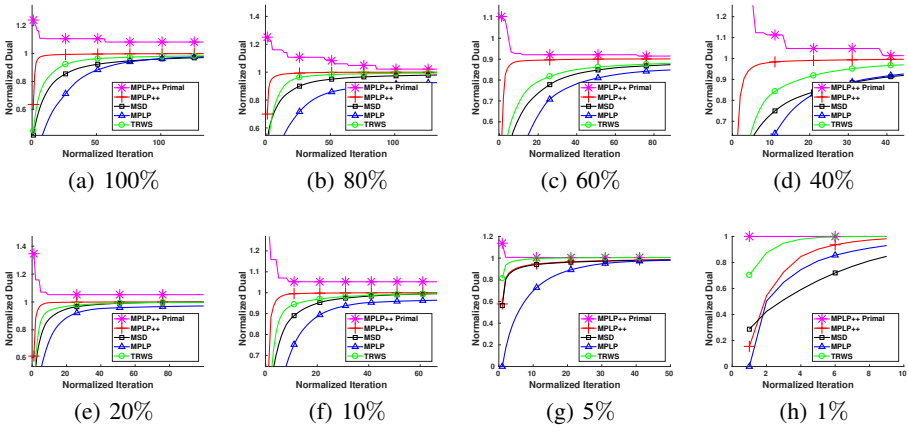
(a) Worms (b) Pose (c) Color-Seg

Fig. A.1. Dual vs Normalized Iterations on three other datasets. The experimental setup and notation is the same as in Fig. 4. Problems in cases (a) and (b) have dense graphs where `MPLP++` outperforms all other algorithms by a substantial margin. In the case (c) graphs are sparse and `TRWS` is dominant.



(a) Worms (b) Pose (c) Color-Seg

Fig. A.2. Dual vs Time (Single Threaded): Fig. shows dual as a function of time for the single threaded versions of all the algorithms. Following the pattern in Fig. A.1 for dense graphs (a) and (b) `MPLP++` dominates all other algorithms by a considerable margin. For sparse graphs, (c) `TRWS` is the fastest. Both the dual $D(\phi)$ and time have been averaged over the entire dataset and normalized to 1.The curves have been normalized such that each instance of the dataset is weighed equally.



(a) 100% (b) 80% (c) 60% (d) 40%

(e) 20% (f) 10% (g) 5% (h) 1%

Fig. A.3. Degradation With Sparsity (Dual vs Iterations): (a)-(h) show graphs with decreasing average connectivity given as percentage of possible edges in figure subcaption. In (a)-(f) `MPLP++` outperforms `TRWS`. Handshake is resilient to graph sparsification even when 90% of the edges have been removed. For (g) and(h) `TRWS` outperforms handshake.

B Formal Proofs

B.1 Additional Notation

To reduce clutter in the proofs we define relevant short-forms here. The set $[n]$ is the set of the first n natural numbers, *i.e.* $[n] = \{1, \dots, n\}$. $[m][n]$ denotes the Cartesian product $[m] \times [n]$. We denote the same reparametrized unaries of two different algorithms $\mathcal{A}$ and $\mathcal{B}$ as $\theta_u^{\mathcal{A}}$ and $\theta_u^{\mathcal{B}}$. $\theta \in \mathbb{R}^{\mathcal{I}}$ is the vector stacked up of unary θ_u and pairwise θ_{uv} potentials. Let $\mathcal{O}_u^{\varepsilon}(\theta) = \{s \mid \theta_u(s) \leq \min_{s'} \theta_u(s') + \varepsilon\}$ be the set of labellings within $\varepsilon > 0$ of the optimal θ_u. $\mathcal{O}_{uv}^{\varepsilon}(\theta)$ is similarly defined for θ_{uv}. Let also $\mathcal{O}^{\varepsilon}(\theta) = \{\mathcal{O}_u^{\varepsilon}(\theta) \mid \forall u \in \mathcal{V}\} \cup \{\mathcal{O}_{uv}^{\varepsilon}(\theta) \mid \forall uv \in \mathcal{E}\}$.

Definition 1. *Tolerance factor ε is the minimum value for which $\mathcal{O}^{\varepsilon}$ contains a consistent labelling,* i.e. *one with node-edge agreement.*

ε is a function of θ and will be written sometimes as $\varepsilon(\theta)$. $\mathcal{O}^{\varepsilon}(\theta)$ can also be thought of as the subset of unary and pairwise labels that are within ε of the optimal labelling.

$\mathcal{O}_u^0(\theta) = \{s \mid \theta_u(s) = \min_{s' \in \mathcal{Y}} \theta_u(s')\}$, then represents the set of optimal labellings. $\mathcal{O}_{uv}^0(\theta)$ can be similarly defined for pairwise potentials θ_{uv}. Then, $\mathcal{O}^0(\theta) = \{\mathcal{O}_u^0(\theta) \mid u \in \mathcal{V}\} \cup \{\mathcal{O}_{uv}^0(\theta) \mid uv \in \mathcal{E}\}$.

The `MPLP++` operator $\mathcal{H}$ can act both on g_{uv} like pairwise costs and on the entire set of costs $\theta \in \mathbb{R}^{\mathcal{I}}$. In the former case, it is exactly as defined in Eqn. ($\mathcal{H}$). The latter case corresponds to an iteration of $\mathcal{H}$. The i-times composition operation of $\mathcal{H}$ on θ denotes i iterations of $\mathcal{H}$ on θ, *i.e.* $\underbrace{\mathcal{H}(\mathcal{H}(\dots(\mathcal{H}(\theta))))}_{\text{i times}} = \mathcal{H}^i(\theta)$. Which is the case would be clear

from the argument the $\mathcal{H}$ operator takes. Also, to denote the resulting cost vector after i iterations of $\mathcal{H}$ on θ, we use notation $\theta^i = \mathcal{H}^i(\theta) = \mathcal{H}(\theta^{i-1})$.

Let $\Theta_{un}(\phi, \kappa)$ be a function that measures the max absolute difference between unary reparameterizations θ^ϕ and θ^κ, defined by $\Theta_{un}(\phi, \kappa) = \max_{u \in \mathcal{V}, s \in \mathcal{Y}} |\theta^\phi_u(s) - \theta^\kappa_v(s)|$. Likewise for pairwise costs, $\Theta_{pw}(\phi, \kappa) = \max_{uv \in \mathcal{E}, st \in \mathcal{Y}^2} |\theta^\phi_{uv}(s,t) - \theta^\kappa_{uv}(s,t)|$.

We also assume that both unary and pairwise costs have been normalized, *i.e.* $\min_{s \in \mathcal{Y}} \theta_u(s) = 0$ and $\min_{st \in \mathcal{Y}^2} \theta_{uv}(s,t) = 0$. A consequence of normalization is $\theta_u \geq 0$ and $\theta_{uv} \geq 0$. The subtracted cost does not affect the labelling and is added to a constant term.

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

Proof. $\boldsymbol{\mathcal{H} \geq \mathcal{M}}$: Let $g_{uv} = \theta_u + \theta_v + \theta_{uv}$. Consider the first line of $(\mathcal{H})$.

$$\theta^{\mathcal{H}}_u(s) := \theta^{\mathcal{M}}_u(s), \quad \theta^{\mathcal{H}}_v(s) := \theta^{\mathcal{M}}_v(s), \ \forall s \in \mathcal{Y} \tag{1}$$

The first line assigns to the reparameterized MPLP++ unaries, $(\theta^{\mathcal{H}}_u, \theta^{\mathcal{H}}_v)$ the unaries resulting from applying the MPLP update to g_{uv}. At this point we have $\mathcal{H} = \mathcal{M}$.
Now, consider the subsequent equations of $\mathcal{H}$

$$\theta^{\mathcal{H}}_v(t) := \theta^{\mathcal{H}}_v(t) + \min_{s \in \mathcal{Y}}[g_{uv}(s,t) - \theta^{\mathcal{H}}_v(t) - \theta^{\mathcal{H}}_u(s)], \ \forall t \in \mathcal{Y}$$
$$\theta^{\mathcal{H}}_u(s) := \theta^{\mathcal{H}}_u(s) + \min_{t \in \mathcal{Y}}[g_{uv}(s,t) - \theta^{\mathcal{H}}_v(t) - \theta^{\mathcal{H}}_u(s)], \ \forall s \in \mathcal{Y}\,.$$

To $\theta^{\mathcal{H}}_v(t)$, a non-negative quantity $\min_{s \in \mathcal{Y}}[g_{uv}(s,t) - \theta^{\mathcal{H}}_v(t) - \theta^{\mathcal{H}}_u(s)]$ is added. Similarly, for $\theta^{\mathcal{H}}_u$. Thus, $\mathcal{H} \geq \mathcal{M}$.

$\boldsymbol{\mathcal{M} \geq \mathcal{U}}$:$\theta^{\mathcal{M}}_u(s) = \min_{t'} g_{uv}(s,t') \geq \min_{s',t'} g_{uv}(s',t') = \theta^{\mathcal{U}}_u(s)$. Likewise, $\theta^{\mathcal{M}}_v(t) = \min_{s'} g_{uv}(s',t) \geq \min_{s',t'} g_{uv}(s',t') = \theta^{\mathcal{U}}_v(t)$. Thus, $\mathcal{M} \geq \mathcal{U}$. □

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

Proof. Let $\theta_u \geq \theta'_u$ and $\theta_v \geq \theta'_v$. We define $g_{uv} = \theta_u + \theta_v + \theta_{uv}$ and $g'_{uv} = \theta'_u + \theta'_v + \theta_{uv}$. Thus we have $g_{uv} \geq g'_{uv}$.

$\mathcal{U}$ is monotonous:
Performing update $\gamma_{\mathcal{U}}[g_{uv}] \rightarrow (\theta^{\mathcal{U}}_u, \theta^{\mathcal{U}}_v)$ (Eqn. $(\mathcal{U})$) yields

$$\theta^{\mathcal{U}}_u(s) = \theta^{\mathcal{U}}_v(t) := \min_{s',t'} \frac{1}{2}\, g_{uv}(s',t') \tag{2}$$

Likewise $\gamma_{\mathcal{U}}[g'_{uv}] \rightarrow (\theta'^{\mathcal{U}}_u, \theta'^{\mathcal{U}}_v)$ yields

$$\theta'^{\mathcal{U}}_u(s) = \theta'^{\mathcal{U}}_v(t) := \min_{s',t'} \frac{1}{2}\, g'_{uv}(s',t') \tag{3}$$

As $g_{uv} \geq g'_{uv}$, we have $\min_{s't'}\ g_{uv}(s',t') \geq \min_{s't'}\ g'_{uv}(s',t')$. This implies, $\theta^{\mathcal{U}}_u \geq \theta'^{\mathcal{U}}_u, \quad \theta^{\mathcal{U}}_v \geq \theta'^{\mathcal{U}}_v$. Hence proved.

$\mathcal{M}$ is monotonous:
Performing update $\gamma_{\mathcal{M}}[g_{uv}] \rightarrow (\theta^{\mathcal{M}}_u, \theta^{\mathcal{M}}_v)$ (Eqn. $\mathcal{M}$) yields

$$\theta_u^{\mathcal{M}}(s) := \frac{1}{2} \min_{t'} \; g_{uv}(s,t'), \quad \theta_v^{\mathcal{M}}(t) := \frac{1}{2} \min_{s'} \; g_{uv}(s',t)$$

For $\gamma_{\mathcal{M}}[g'_{uv}] \rightarrow (\theta_u'^{\mathcal{M}}, \theta_v'^{\mathcal{M}})$ yields

$$\theta_u'^{\mathcal{M}}(s) := \frac{1}{2} \min_{t'} \; g'_{uv}(s,t'), \quad \theta_v'^{\mathcal{M}}(t) := \frac{1}{2} \min_{s'} \; g'_{uv}(s',t)$$

As $g_{uv} \geq g'_{uv}$, we have $\min_{t'} \; g_{uv}(s,t') \geq \min_{t'} \; g'_{uv}(s,t') \;\; \& \;\; \min_{s'} \; g_{uv}(s',t) \geq \min_{s'} \; g'_{uv}(s',t)$. This implies $\theta_u^{\mathcal{M}} \geq \theta_u'^{\mathcal{M}}, \;\; \theta_v^{\mathcal{M}} \geq \theta_v'^{\mathcal{M}}$. Hence proved.

$\mathcal{H}$ is not monotonous:

To prove that $\mathcal{H}$ is not monotonous we show a counter-example to the monotonous condition stated in theorem 3.

Consider the following unary and pairwise costs,

$$\theta_u = \begin{pmatrix} 4 \\ 0 \end{pmatrix} \;\; \theta_v = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \;\; \theta'_u = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \;\; \theta'_v = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \;\; \theta_{uv} = \begin{pmatrix} 0 & 1 \\ 7 & 5 \end{pmatrix}$$

These costs satisfy $\theta_u \geq \theta'_u$ and $\theta_v \geq \theta'_v$.

Now, applying the $\mathcal{H}$ operation to g_{uv} and g'_{uv}, we get the following potentials

$$\theta_u^{\mathcal{H}} = \begin{pmatrix} 2.5 \\ 2.5 \end{pmatrix} \;\; \theta_v^{\mathcal{H}} = \begin{pmatrix} 3.5 \\ 2.5 \end{pmatrix} \theta_u'^{\mathcal{H}} = \begin{pmatrix} 0 \\ 4 \end{pmatrix} \;\; \theta_v'^{\mathcal{H}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

We thus have $\theta_u^{\mathcal{H}} \not\geq \theta_u'^{\mathcal{H}}$, providing the necessary counter-example. □

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.*

Proof. From the definition of dominance we have, if γ dominates μ ($\gamma \geq \mu$) then $\gamma[g_{uv}] \geq \mu[g_{uv}]$, $\forall g_{uv}$.

By the definition of *monotonous*, if μ is monotonous, $g^1_{uv} \geq g^2_{uv} \implies \mu[g^1_{uv}] \geq \mu[g^2_{uv}]$.

Now during the 1^{st} iteration we have three cases,

- Case **A**: Nodes u and v of edge uv have not been reparametrized before.
- Case **B**: Nodes u and v of edge uv have been reparametrized before.
- Case **C**: Only node u or v of edge uv have been reparametrized before.

If case **A** we have by dominance $\gamma[g_{uv}] \geq \mu[g_{uv}]$.

If case **B**, the unaries u and v have been reparametrized. Let the reparametrized unaries for γ be $(\theta^{\gamma}_u, \theta^{\gamma}_v)$ and for μ be $(\theta^{\mu}_u, \theta^{\mu}_v)$. From $\gamma \geq \mu$ we know $\theta^{\gamma}_u \geq \theta^{\mu}_u$ and

$\theta_v^\gamma \geq \theta_v^\mu$. Then, $g_{uv}^\gamma = \theta_u^\gamma + \theta_v^\gamma + \theta_{uv}$ and $g_{uv}^\mu = \theta_u^\mu + \theta_v^\mu + \theta_{uv}$. It follows $g_{uv}^\gamma \geq g_{uv}^\mu$. Consider the chain of inequalities

$$\gamma[g_{uv}^\gamma] \geq \mu[g_{uv}^\gamma] \geq \mu[g_{uv}^\mu] \tag{4}$$

The first is true by $\gamma \geq \mu$ dominance. The second is true by μ-monotonicity. Thus for case B also, γ results in reparametrized unaries that are co-ordinate wise greater than μ.

Case **C**, can be proven in much the same way as case B.

□

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

To prove node-edge agreement we have to show that

$$\lim_{i \to \infty} \varepsilon(\mathcal{H}^i(\theta)) = 0 \tag{5}$$

where ε is the tolerance factor defined in § B.1.

By saying that MPLP++ converges to node-edge agreement, we mean that as the algorithm progresses ε tends to 0 and the set of labels belonging to $\mathcal{O}^\varepsilon$ is sequentially pruned only to leave an optimal labelling satisfying the node-edge agreement condition, converting $\mathcal{O}^\varepsilon$ to $\mathcal{O}^0$.

The proof is dependent on several lemmas which are sequentially proved.

Lemma B.1. *$\mathcal{H}$ is a continuous function.*

Proof. Stated differently, we show the proposed reparameterizations in this paper are continuous. To generalize the result across all proposed reparameterizations we recall the idea of an *oracle call* from the main paper, *i.e.* operations of the type $\min_{t \in \mathcal{Y}} g_{uv}(s,t)$, $\forall s \in \mathcal{Y}$. We show the continuity of the first equation of the MPLP++ operation only (1^{st} equation of Eqn. ($\mathcal{H}$) in the main paper). The other equations of MPLP++ and other algorithms can be proved similarly.

We prove continuity for only one label of one unary cost (label s and unary u). Let $\delta > 0$. Similar proofs hold for all other unaries and pairwise costs. Consider two edges denoted by the triplet $(\theta_u, \theta_v, \theta_{uv})$ and $(\theta'_u, \theta_v, \theta_{uv})$, such that

$$|\theta'_u(s) - \theta_u(s)| < \delta \tag{6}$$

Then, $g_{uv} := \theta_u + \theta_v + \theta_{uv}$ and $g'_{uv} := \theta'_u + \theta_v + \theta_{uv}$. By (6), we have

$$\forall t \;\; |g'_{uv}(s,t) - g_{uv}(s,t)| < \delta \tag{7}$$

Applying the first operation of MPLP++ to g_{uv} and g'_{uv}, we get

$$\theta_u^{\mathcal{H}}(s) := \frac{1}{2} \min_{t \in \mathcal{Y}} g_{uv}(s,t) \tag{8}$$

$$\theta_u'^{\mathcal{H}}(s) := \frac{1}{2} \min_{t \in \mathcal{Y}} g'_{uv}(s,t) \tag{9}$$

Let $\mid \theta'^{\mathcal{H}}_u(s) - \theta^{\mathcal{H}}_u(s) \mid < \nu$, then by choosing $\delta := \nu/2$, we have

$$\forall \theta \in \mathbb{R}^{\mathcal{I}},\ \forall \nu > 0, \exists \delta > 0,\ \forall \theta' \in \mathbb{R}^{\mathcal{I}},\ \mid \theta'_u(s) - \theta_u(s) \mid < \delta \implies \mid \theta'^{\mathcal{H}}_u(s) - \theta^{\mathcal{H}}_u(s) \mid < \nu \tag{10}$$

□

Lemma B.2. *The dual function D is a continuous function in it's input θ.*

Proof. Consider two different reparameterizations $\theta^\phi, \theta^\kappa \in \mathbb{R}^{\mathcal{I}}$. To prove $D(.)$ is a continuous function, it suffices to show $\forall \nu > 0, \exists \delta > 0$, such that $|\theta^\phi - \theta^\kappa| < \delta \implies |D(\theta^\phi) - D(\theta^\kappa)| < \nu$. To do so, we need to recall from § B.1, $\Theta_{pw}(\phi, \kappa) = \max_{uv \in \mathcal{E}, st \in \mathcal{Y}^2} |\theta^\phi_{uv}(s,t) - \theta^\kappa_{uv}(s,t)|$ and $\Theta_{un}(\phi, \kappa) = \max_{u \in \mathcal{V}, s \in \mathcal{Y}} |\theta^\phi_u(s) - \theta^\kappa_u(s)|$.

We then have

$$\begin{aligned}
\mid D(\theta^\phi) - D(\theta^\kappa) \mid &= \mid (\sum_{u \in \mathcal{V}} \min_s \theta^\phi_u(s) + \sum_{uv \in \mathcal{E}} \min_{(s,t)} \theta^\phi_{uv}(s,t)) - (\sum_{u \in \mathcal{V}} \min_s \theta^\kappa_u(s) + \sum_{uv \in \mathcal{E}} \min_{s,t} \theta^\kappa_{uv}(s,t)) \mid \\
&= \mid \sum_{u \in \mathcal{V}} (\min_s \theta^\phi_u(s) - \min_s \theta^\kappa_u(s)) + \sum_{uv \in \mathcal{E}} (\min_{s,t} \theta^\phi_{uv}(s,t)) - \min_{s,t} \theta^\kappa_{uv}(s,t)) \mid \\
&\leq \mid \sum_{u \in \mathcal{V}} (\min_s \theta^\phi_u(s) - \min_s \theta^\kappa_u(s)) \mid + \mid \sum_{uv \in \mathcal{E}} (\min_{s,t} \theta^\phi_{uv}(s,t)) - \min_{s,t} \theta^\kappa_{uv}(s,t)) \mid \\
&\leq \mid \sum_{u \in \mathcal{V}} \Theta_{un}(\phi, \kappa) \mid + \mid \sum_{uv \in \mathcal{E}} \Theta_{pw}(\phi, \kappa) \mid \leq \mid \mathcal{V} \mid\mid \Theta_{un}(\phi, \kappa) \mid + \mid \mathcal{E} \mid\mid \Theta_{pw}(\phi, \kappa) \mid
\end{aligned}$$

Thus for $|D(\theta^\phi) - D(\theta^\kappa)| < \nu$, we need to choose a δ as a function of $|\Theta_{pw}| < \frac{1}{|\mathcal{E}|}$ and $|\Theta_{un}| < \frac{1}{|\mathcal{V}|}$. This can be done as $\mathcal{H}$ is continuous (by proposition B.1) and pointwise-maxima are also continuous [53]. □

Lemma B.3. *Tolerance factor ε is continuous on θ.*

Proof. Let θ^ϕ and θ^κ be two reparameterizations. We prove ε is continuous for unaries only. The proof for pairwise terms is similar. Let $\Theta_{un}(\phi, \kappa) = \delta$. From the definition of Θ_{un} we have $\max_{u \in \mathcal{V}, s \in \mathcal{Y}} |\theta^\phi_u(s) - \theta^\kappa_u(s)| = \delta$. Thus,

$$|\theta^\phi_u(s) - \theta^\kappa_u(s)| \leq \delta \tag{11}$$

(11) can be rewritten as

$$\theta^\kappa_u(s) - \delta \leq \theta^\phi_u(s) \leq \theta^\kappa_u(s) + \delta \tag{12}$$

Now consider the set $\mathcal{O}^\varepsilon_u(\theta) = \{s | \theta_u(s) \leq \min_{s'}(\theta_u(s') + \varepsilon(\theta))\}$. For a unary in $\mathcal{O}^\varepsilon_u(\theta)$

$$\theta^\phi_u(s) \leq \min_{s'}(\theta^\phi_u(s') + \varepsilon(\theta)) \tag{13}$$

Substituting (12) in (13) we get,

$$\theta_u^\kappa(s) - \delta \leq \min_{s'}(\theta_u^\kappa(s') + \delta + \varepsilon(\theta)) \implies \theta_u^\kappa(s) \leq \min_{s'}(\theta_u^\kappa(s') + \varepsilon(\theta) + 2\delta) \quad (14)$$

Thus if s satisfies (13), it also satisfies (14), which has $\varepsilon' \geq \varepsilon + 2\delta$. □

Lemma B.4. $D(\mathcal{H}^{i+1}(\theta)) \geq D(\mathcal{H}^i(\theta)), \forall i,$ i.e. *the* MPLP++ *reparametrization never decreases the dual* D.

Proof. Let's consider D to be fixed for all variables except the block D_{uv}. Also, we assume that the costs have been normalized, *i.e.* $\min_{s\in\mathcal{Y}} \theta_u(s) = 0$, $\min_{t\in\mathcal{Y}} \theta_v(t) = 0$ and $\min_{st\in\mathcal{Y}^2} \theta_{uv}(s,t) = 0$. So, we have $D_{uv}(\theta) = 0$. We thus have to show that $D_{uv}(\mathcal{H}(\theta)) \geq D_{uv}(\theta) = 0$.

The aggregated potential $g_{uv} = \theta_u + \theta_v + \theta_{uv}$ can be written in the form of a $\mathcal{Y} \times \mathcal{Y}$ matrix

$$\begin{bmatrix} r_1 + \Delta_{1,1} & \dots & r_1 + \Delta_{1,|\mathcal{Y}|} \\ \vdots & \ddots & \vdots \\ r_{|\mathcal{Y}|} + \Delta_{|\mathcal{Y}|,1} & \dots & r_{|\mathcal{Y}|} + \Delta_{|\mathcal{Y}|,|\mathcal{Y}|} \end{bmatrix}$$

where r_s is the *row minimum* and $\Delta_{s,t} \geq 0$. So, $\Delta_{s,t}$ is 0 for all elements of the row that are equal to r_s. As each element of row s of g_{uv} contains $\theta_u(s)$, we know that $r_s \geq \theta_u(s), \forall s \in \mathcal{Y}$.

Now, consider the 1^{st} equation of the MPLP $++$ operation ($\mathcal{H}$)

$$\theta_u^{\mathcal{H}}(s) := \tfrac{1}{2}\min_{t\in\mathcal{Y}}[g_{uv}(s,t)] = \tfrac{1}{2}\min_{t\in\mathcal{Y}} r_s + \Delta_{s,t} = \tfrac{r_s}{2},\ \forall s \in \mathcal{Y}\,,$$
$$\theta_v^{\mathcal{H}}(t) := \tfrac{1}{2}\min_{s\in\mathcal{Y}}[g_{uv}(s,t)] = \tfrac{1}{2}\min_{s\in\mathcal{Y}} r_s + \Delta_{s,t},\ \forall t \in \mathcal{Y}$$

It is $r_s \geq \theta_u(s) \geq 0$ and therefore, $\theta_u^{\mathcal{H}}(s) = r_s/2 \geq 0$. Likewise, $\theta_v^{\mathcal{H}}(t) = \frac{1}{2}\min_{s\in\mathcal{Y}}[r_s + \Delta_{s,t}]$, where $r_s \geq 0$ and $\Delta_{s,t} \geq 0\ \forall t$, thus $\theta_v^{\mathcal{H}}(t) \geq 0$. Hence, $D(\mathcal{H}(\theta)) \geq 0$. □

Lemma B.5. $\mathcal{H}$ *converges to a fixed point.*

Proof. Initially, the dual bound $D(\theta^\phi)$ is computed using (3) which takes the min over all unaries $\theta_u(s)$ and pairwise terms $\theta_{uv}(s,t)$ individually. Thus, $D(\theta^\phi)$ is bounded unless for any $\theta_u(s)$ or $\theta_{uv}(s,t)$ all the elements are ∞. If the dual is unbounded, then by strong duality [53] the primal is also unbounded, and the only energy attainable is ∞. On the other hand, if $D(\theta^\phi)$ is bounded, by the LP duality theorem, the primal energy $E(y|\theta)$ serves as an upper bound for the $D(\theta^\phi)$.

We have proven that the BCA-update $\mathcal{H}$ brings about a monotonic improvement of $D(\theta^\phi)$. As each algorithm involves performing these updates over a sequence of edges e^i covering the entire graph, we end up at the end of every iteration with a non-decreasing dual $D(\theta^\phi)$. Thus, each algorithm generates an increasing sequence of $D(\theta^\phi)$ which is bounded from above and by the *Monotone Convergence Theorem* for real numbers $\mathbb{R}$ [54] (Section 3.3) the algorithm converges. □

Lemma B.6. *$\mathcal{H}$ is bounded w.r.t. D: For any θ there exists an $M \in \mathbb{R}$ such that $D(\mathcal{H}^i(\theta)) < M$ for any i.*

Proof. As D is a dual LP, we know from LP-Duality Theorem [53] that $D(\mathcal{H}^i(\theta)) \leq E(y|\theta)$, where E is as in (1) and y is the labelling. Also, an alternative upper bound can be constructed as follows. Let $M_u = \max_s \theta_u^{\mathcal{H}}(s)$, $M_{uv} = \max_{st} \theta_{uv}^{\mathcal{H}}(s,t)$. We then have $|\theta_u^{\mathcal{H}}(s)| \leq M_u, \forall s$ and $|\theta_{uv}^{\mathcal{H}}(s,t)| \leq M_{uv}\ \forall s,t$. This also implies $\min_{s\in\mathcal{Y}} \theta_u^{\mathcal{H}}(s) \leq M_u$ and $\min_{st\in\mathcal{Y}^2} \theta_{uv}^{\mathcal{H}}(s,t) \leq M_{uv}$.

Thus we have

$$D(\mathcal{H}(\theta)) = \sum_{u\in\mathcal{V}} \min_s \theta_u^{\mathcal{H}}(s) + \sum_{uv\in\mathcal{E}} \min_{s,t} \theta_{uv}^{\mathcal{H}}(s,t) \ \leq \sum_{u\in\mathcal{V}} M_u + \sum_{uv\in\mathcal{E}} M_{uv}$$

Therefore $D(\mathcal{H}(\theta))$ is always bounded by $M_{\mathcal{H}(\theta)} = \sum_{u\in\mathcal{V}} M_u + \sum_{uv\in\mathcal{E}} M_{uv}$. We know from Lemma B.5 that $\mathcal{H}$ converges to a fixed point, thus after a certain number of iterations no changes occur in $\mathcal{H}^i(\theta)$. To compute the required bound, one must simply take $\max_i M_{\mathcal{H}^i(\theta)}$ over all reparameterizations that have occurred. □

Lemma B.7. *For any θ there exists $C > 0$ such that $||\mathcal{H}^i(\theta)|| \leq C||\theta||$ for any i.*

Proof. We start off by showing that the lemma is true for one label of a unary. Let $\theta^\phi(s)$ be the reparameterization of $\theta_u(s)$. Consider the edge triplet $(\theta_u, \theta_v, \theta_{uv})$. Let $m_u := \min_{s\in\mathcal{Y}} \theta_u(s)$, $M_u := \max_{s\in\mathcal{Y}} \theta_u(s)$, $M_v := \max_{t\in\mathcal{Y}} \theta_v(t)$ and $M_{uv} := \max_{st\in\mathcal{Y}^2} \theta_{uv}(s,t)$.

$$\begin{aligned}
\theta_u^{\mathcal{H}}(s) &:= \frac{1}{2}\min_t g_{uv}(s,t) \\
\theta_u^{\mathcal{H}}(s) &:= \frac{1}{2}\theta_u(s) + \frac{1}{2}\min_t\{\theta_v(t) + \theta_{uv}(s,t)\} \\
\theta_u^{\mathcal{H}}(s) &\leq \frac{1}{2}\theta_u(s) + \frac{1}{2}(M_v + M_{uv}) \\
\theta_u^{\mathcal{H}}(s) &\leq \left(1 + \frac{M_v + M_{uv}}{\theta_u(s)}\right)\theta_u(s) \leq \left(1 + \frac{M_v + M_{uv}}{m_u}\right)\theta_u(s)
\end{aligned}$$

Thus, we can choose $C_u^i = (1 + \frac{M_u + M_{uv}}{m_u})$ and get a bound $||\theta_u^{\mathcal{H}}|| \leq C_u^i||\theta_u||$. As we have to find an upper bound $\forall\theta_u$ and $\forall\theta_{uv}$, we repeat the process and then take the max over all C_u^i and C_{uv}^i obtaining C^i. This gives us a C^i that satisfies $||\theta^{i+1} = \mathcal{H}(\theta^i)|| \leq C^i||\theta^i||$. By Lemma B.5 after a certain number of iterations n, θ^n converges, so to get the required result we simply simply take $C = \prod_{i=1}^n C^i$, giving $||\mathcal{H}^i(\theta)|| \leq C||\theta||$ for any i. □

Lemma B.8. *$D(\mathcal{H}(\theta)) = D(\theta)$ implies $\mathcal{O}^0(\mathcal{H}(\theta)) \subseteq \mathcal{O}^0(\theta)$,* i.e. *at a fixed point no new optimal labellings are achieved. If additionally $\varepsilon(\theta^{\mathcal{H}}) > 0$, then $\mathcal{O}^0(\mathcal{H}(\theta)) \subset \mathcal{O}^0(\theta)$,* i.e. *the inequality is strict.*

Proof. Let $\theta^{\mathcal{H}} = \mathcal{H}(\theta)$, *i.e.* $\theta^{\mathcal{H}}$ is the reparameterized cost vector obtained after reparameterizing the original cost vector θ. We prove the first assertion for unary cost u only, the remaining costs can be proved similarly. To do so, we assume that only after reparameterization only the optimal label of unary cost u has changed.

Let this label be $s^* = \arg\min_s \theta^{\mathcal{H}}_u(s)$. This implies $s^* \in \mathcal{O}(\theta^{\mathcal{H}})$. Now, to prove the first assertion, we have to show the inclusion $s^* \in \mathcal{O}^0_u(\theta^{\mathcal{H}}) \implies s^* \in \mathcal{O}^0_u(\theta)$.

Let us additionally define a function η that takes as input a node index p and outputs its optimal labelling, *i.e.* $k = \eta(v)$ implies $k = \arg\min_{s\in\mathcal{Y}} \theta_p(s)$. We define this function over all unary costs except u. As $\theta^{\mathcal{H}}$ and θ differ in only the label u, we can use η for choosing the optimal labelling for both of them.

Since, $D(\theta^{\mathcal{H}}) = D(\theta)$, we have

$$\sum_{v\in\mathcal{V}} \min_{s\in\mathcal{Y}} \theta^{\mathcal{H}}_v(s) + \sum_{vw\in\mathcal{E}} \min_{st\in\mathcal{Y}^2} \theta^{\mathcal{H}}_{vw}(s,t) = \sum_{v\in\mathcal{V}} \min_{s\in\mathcal{Y}} \theta_v(s) + \sum_{vw\in\mathcal{E}} \min_{st\in\mathcal{Y}^2} \theta_{vw}(s,t) \quad (15)$$

Since all the labels but the ones for u are the same, the terms on both sides of (15) cancel out leaving

$$\min_{s\in\mathcal{Y}} \theta^{\mathcal{H}}_u(s) + \sum_{uv|v\in Nb(u)} \min_{st\in\mathcal{Y}^2} \theta^{\mathcal{H}}_{uv}(s,t) = \min_{s\in\mathcal{Y}} \theta_u(s) + \sum_{uv|v\in Nb(u)} \min_{st\in\mathcal{Y}^2} \theta_{uv}(s,t) \quad (16)$$

Substituting $s^* = \arg\min_s \theta^{\mathcal{H}}_u(s)$ and η into (16) we get

$$\theta^{\mathcal{H}}_u(s^*) + \sum_{uv|v\in Nb(u)} \theta^{\mathcal{H}}_{uv}(s^*, \eta(v)) = \min_{s\in\mathcal{Y}} \theta_u(s) + \sum_{uv|v\in Nb(u)} \theta_{uv}(s, \eta(v)) \quad (17)$$

Now, by the definition of reparameterization we have, $\forall s \in \mathcal{Y}$

$$\theta^{\mathcal{H}}_u(s) + \sum_{uv|v\in Nb(u)} \theta^{\mathcal{H}}_{uv}(s, \eta(v)) = \theta_u(s) + \sum_{uv|v\in Nb(u)} \theta_{uv}(s, \eta(v)) \quad (18)$$

This also holds true for label s^*, thus

$$\theta^{\mathcal{H}}_u(s^*) + \sum_{uv|v\in Nb(u)} \theta^{\mathcal{H}}_{uv}(s^*, \eta(v)) = \theta_u(s^*) + \sum_{uv|v\in Nb(u)} \theta_{uv}(s^*, \eta(v)) \quad (19)$$

Now, equating the RHS of (19) and the RHS of (17), we get

$$\theta_u(s^*) + \sum_{uv|v\in Nb(u)} \theta_{uv}(s^*, \eta(v)) = \min_{s\in\mathcal{Y}} \theta_u(s) + \sum_{uv|v\in Nb(u)} \theta_{uv}(s, \eta(v)) \quad (20)$$

Thus $s^* = \arg\min_{s\in\mathcal{Y}} \theta_u(s) \implies s^* \in \mathcal{O}^0_u(\theta)$, proving the first assertion.

To prove the second assertion, we have to show that if $\epsilon(\theta^{\mathcal{H}}) > 0$ and $D(\mathcal{H}(\theta)) = D(\theta)$, there exists $s' \in \mathcal{O}^0_u(\theta)$ such that $s' \notin \mathcal{O}^0_u(\theta^{\mathcal{H}})$.

Since, $s' \in \mathcal{O}_u^0(\theta)$, we have

$$\theta_u(s') + \sum_{v \in Nb(u)} \theta_{uv}(s', \eta(v)) = \min_{s \in \mathcal{Y}} \theta_u(s) + \sum_{v \in Nb(u)} \min_{s \in \mathcal{Y}} \theta_{uv}(s, \eta(v)) \tag{21}$$

From the reparameterization relation we have

$$\theta_u(s') + \sum_{v \in Nb(u)} \theta_{uv}(s', \eta(v)) = \theta_u^{\mathcal{H}}(s') + \sum_{v \in Nb(u)} \theta_{uv}^{\mathcal{H}}(s', \eta(v)) \tag{22}$$

Since $\varepsilon(\theta^{\mathcal{H}}) > 0$, it has to be the case that

$$\theta_u^{\mathcal{H}}(s') + \sum_{v \in Nb(u)} \theta_{uv}^{\mathcal{H}}(s', \eta(v)) > \min_{s \in \mathcal{Y}} \theta_u^{\mathcal{H}}(s) + \sum_{v \in Nb(u)} \min_{s \in \mathcal{Y}} \theta_{uv}^{\mathcal{H}}(s, \eta(v)) \tag{23}$$

otherwise $\varepsilon(\theta^{\mathcal{H}}) = 0$. Thus, $s' \notin \mathcal{O}_u^0(\theta^{\mathcal{H}})$.

Lemma B.9. *There exists an n, such that $D(\mathcal{H}^{n+m}(\theta)) = D(\mathcal{H}^n(\theta))$ implies $\mathcal{O}^0(\mathcal{H}^{n+m}(\theta)) = \mathcal{O}^0(\mathcal{H}^n(\theta)), \forall m \geq 0$.*

Proof. Let $\theta^0, \theta^1, ...$ be the sequence of vectors generated from $\mathcal{H}$ via $\theta^{i+1} = \mathcal{H}(\theta^i)$. After a certain number of iterations, we have to show that $\mathcal{O}^0(\theta^{n+m}) = \mathcal{O}^0(\theta^n)$, for all $m \geq 0$. From Lemma B.8 we have after a fixed point has been reached at iteration i, $\mathcal{O}^0(\theta^i) \supset \mathcal{O}^0(\theta^{i+1}) \supset \mathcal{O}^0(\theta^{i+2}) \supset ....$ Since $\mathcal{O}^0(\theta)$ is finite, it cannot shrink indefinitely and after a certain number (n) of iterations $\mathcal{O}^0(\theta^n)$ will become consistent. Yielding, $\mathcal{O}^0(\theta^{m+n}) = \mathcal{O}^0(\theta^n), \forall m \geq 0$. □

Now, we are ready to prove Theorem 1

Combining the above lemmas, we introduce the notion of *consistency-enforcing* algorithms. An algorithm is consistency-enforcing if it satisfies Lemmas B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, .

The `MPLP++` operator $\mathcal{H}$ is consistency enforcing then $\lim_{i \to \infty} \varepsilon(\mathcal{H}^i(\theta)) = 0$.

$$\lim_{i \to \infty} \varepsilon(\mathcal{H}^i(\theta)) = 0 \tag{24}$$

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

Proof. By virtue of Lemma B.7 and Lemma B.4, the sequence $\theta^i = \mathcal{H}^i(\theta)$ is bounded. Therefore, by the Bolzano-Weierstrass Theorem [54], there exists a converging subsequence $\theta^{i(j)}$, $j = 1, 2, \ldots$, where $j > j'$ implies $i(j) > i(j')$, *i.e.* the limit $\theta^* := \lim_{j \to \infty} \theta^{i(j)}$ exists. Let us show that it holds

$$\varepsilon(\theta^*) = 0 \tag{25}$$

for any converging subsequence of θ^i.

Since due to convergence of $\mathcal{H}$ (shown in Lemma B.4) and Lemma B.6 the sequence $D(\theta^i)$ is non-decreasing and bounded from above, and therefore converges to a limit point $D^* := \lim_{i \to \infty} D(\theta^i)$. Therefore, it also holds

$$D^* = \lim_{j\to\infty} D(\theta^{i(j)}) = \lim_{j\to\infty} D(\theta^{i(j)+n}), \quad \forall n \geq 0 \tag{26}$$

This implies

$$0 = \lim_{j\to\infty} D(\theta^{i(j)}) - \lim_{j\to\infty} D(\theta^{i(j)+n}) = \lim_{j\to\infty} D(\theta^{i(j)}) - D(\mathcal{H}^n(\theta^{i(j)})). \tag{27}$$

Since D is continuous it holds

$$0 = \lim_{j\to\infty} (D(\theta^{i(j)}) - D(\mathcal{H}^n(\theta^{i(j)}))). \tag{28}$$

and therefore, (25) holds by virtue of Lemma B.9.

Since, ε is a continuous function, (25) implies

$$\lim_{j\to\infty} \varepsilon(\theta^{i(j)}) = 0 \tag{29}$$

for any converging sub-sequence $\theta^{i(j)}$.

Now, considering the sequence $\varepsilon(\theta^i)$, we know by virtue of Lemma B.1 $s^i := \sup_{j\geq i} \varepsilon(\theta^j)$. Sequence s^i is a monotonically non-increasing sequence of non-negative numbers and therefore it has a limit $s^* = \lim_{i\to\infty} s^i$.

According to the "Theorem of Superior and Inferior Limits" [54] there exists a subsequence $\varepsilon(\theta^{i'(j)})$ such that

$$\lim_{j\to\infty} \varepsilon(\theta^{i'(j(k))}) = s^* \tag{30}$$

The sequence $\theta^{i'(j)}$ is bounded virtue of Lemma B.7 and therefore contains a converging subsequence $\theta^{i'(j(k))}$. For this subsequence it also holds,

$$\lim_{k\to\infty} \varepsilon(\theta^{i'(j(k))}) = s^* \tag{31}$$

At the same time, as proved in (29), for any converging subsequence it holds

$$0 = \lim_{k\to\infty} \varepsilon(\theta^{i'(j(k))}) = s^* = \lim_{i\to\infty} \sup_{k\geq i} \varepsilon(\theta^k) \tag{32}$$

Finally, $0 \leq \varepsilon(\theta^i) \leq \sup_{k\geq i} \varepsilon(\theta^k)$ implies $\lim_{i\to\infty} \varepsilon(\theta^i) = 0$. □

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 $\theta^\phi_{uv}(\cdot,\cdot)$.*

Proof. The if clause has been proven in ([20], Theorem 2). The only if clause can be proven as follows:

As mentioned in the main paper and shown in [15, 21] the Dual LP $D(\phi)$ is a concave, piecewise linear function.

For proving optimality of block $D_{uv}(\phi_{u\leftrightarrow v})$ we need to show $\mathbf{0} \in \partial D_{uv}(\phi_{u\leftrightarrow v})$, where $\partial D_{uv}(\phi_{u\leftrightarrow v})$ is the super-differential of $D_{uv}(\phi_{u\leftrightarrow v})$. Reconsidering the dual $D_{uv}(\phi_{u\leftrightarrow v})$

$$D_{uv}(\phi_{u\leftrightarrow v}) := \min_{st\in\mathcal{Y}^2} \theta^{\phi}_{uv}(s,t) + \min_{s\in\mathcal{Y}} \theta^{\phi}_{u}(s) + \min_{t\in\mathcal{Y}} \theta^{\phi}_{v}(t)\,, \tag{33}$$

Let $x'_u = \arg\min_{s\in\mathcal{Y}} \theta^{\phi}_u(s)$, $x'_v = \arg\min_{t\in\mathcal{Y}} \theta^{\phi}_v(t)$ and $(x''_u, x''_v) = \arg\min_{st\in\mathcal{Y}^2} \theta^{\phi}_{uv}(s,t)$. Taking the super-gradient of $D_{uv}(\phi_{u\leftrightarrow v})$ w.r.t. $\phi_{u\to v}$, we have

$$\frac{\partial D_{uv}(\phi_{u\leftrightarrow v})}{\partial \phi_{u\to v}(s)} :\, = \begin{cases} 0, & s \neq x'_u,\ s \neq x''_u \\ 0, & s = x'_u,\ s = x''_u \\ 1, & s = x'_u,\ s \neq x''_u \\ -1, & s \neq x'_u,\ s = x''_u \end{cases} \tag{34}$$

Likewise, taking the super-gradient of $D_{uv}(\phi_{u\leftrightarrow v})$ w.r.t. $\phi_{v\to u}$, we have

$$\frac{\partial D_{uv}(\phi_{u\leftrightarrow v})}{\partial \phi_{v\to u}(t)} :\, = \begin{cases} 0, & t \neq x'_v,\ t \neq x''_v \\ 0, & t = x'_v,\ t = x''_v \\ 1, & t = x'_v,\ t \neq x''_v \\ -1, & t \neq x'_v,\ t = x''_v \end{cases} \tag{35}$$

Thus, if $s = x'_u = x''_u$ and $t = x'_v = x''_v$, we have $0 \in \frac{\partial D_{uv}(\phi_{v\leftrightarrow u})}{\partial \phi_{v\leftrightarrow u}(s,t)}$, proving the only if clause. □