Taxonomy of Dual Block-Coordinate Ascent Methods for Discrete Energy Minimization

Siddharth Tourani[1] **Alexander Shekhovtsov**[2] **Carsten Rother**[1] **Bogdan Savchynskyy**[1]
[1]University of Heidelberg, Germany [2]Czech Technical University in Prague

Abstract

We consider the maximum-a-posteriori inference problem in discrete graphical models and study solvers based on the dual block-coordinate ascent rule. We map all existing solvers in a single framework, allowing for a better understanding of their design principles. We theoretically show that some block-optimizing updates are sub-optimal and how to strictly improve them. On a wide range of problem instances of varying graph connectivity, we study the performance of existing solvers as well as new variants that can be obtained within the framework. As a result of this exploration we build a new state-of-the art solver, performing uniformly better on the whole range of test instances.

1. INTRODUCTION

Discrete graphical models, one of the most sound and powerful frameworks in computer vision and machine learning, is still used in many applications in the era of CNNs. Graphical models effectively encode domain specific prior information in the form of a structured cost function, which is often hard to learn from data directly. With an increase in parallelization, fast dual block-coordinate ascent algorithms (BCA) have been developed that allow their application *e.g.* in stereo (Shekhovtsov et al., 2016), optical flow (Munda et al., 2017), 6D pose estimation (Tourani et al., 2018). Combined and jointly trained with CNNs they can create more powerful models (Chen et al., 2015; Knöbelreiter et al., 2017). They can also provide efficient regularization for training of CNN models (Kolesnikov and Lampert, 2016; Marin et al., 2019; Tosi et al., 2019). Applications where structural constraints must be fulfilled (*e.g.* (Payer et al., 2016)) or the optimality is required also significantly benefit from fast computation of good lower bounds by such methods (Savchynskyy et al., 2013; Haller et al., 2018).

In this work we systematically review the existing BCA methods. Despite being developed for different dual decompositions, they can be equivalently formulated as BCA methods on the same dual problem. We contribute a theoretical analysis showing which block updates are sub-optimal and can be improved. We perform an experimental study on a corpus of very diverse problem instances to discover important properties relevant to algorithm design. Such as, which types of variable updates are more efficient, or whether a dynamic or static strategy in sub-problem selection is better, *etc*. There is currently no single algorithm that would work well for both, sparse and dense problems. With this new comparison and theoretical insights, we synthesize a novel BCA method, that selects subproblems automatically adapting to the given graph structure. It applies the type of updates that are more expensive but which turn out to be more efficient and performs universally better across the whole range of the problems in the datasets we used.

1.1 Related Work

Inference in graphical models is a well-known NP-hard problem. A number of solvers with different time complexities and guarantees, utilized in different applications, is surveyed in (Kappes et al., 2015; Li et al., 2016). The linear-programming approach and the large family of associated methods is well covered in (Werner, 2007; Savchynskyy, 2019). In this work we focus on BCA methods, which appear to offer the best lower bounds with a limited time budget for pairwise models. These methods can be used to obtain fast approximate solutions directly, or to efficiently reduce the full combinatorial search (Savchynskyy et al., 2013; Haller et al., 2018). Many BCA methods have been proposed to date and we selected in Table 1 a mostly complete and representative list of the state-of-the-art BCA algorithms. Some of these methods were originally obtained for different dual formulations, based on the decompositions into larger subproblems (`TRW-S`, `TBCA`, `DMM`). Although,

Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

Abbreviation	Authors	Method Name	Type of blocks / updates
MSD	Schlesinger and Antoniuk (2011)	Min-Sum Diffusion	Node-adjacent, isotropic
CMP	Hazan and Shashua (2010, Alg.5)	Convex Max Product	
TRW-S	Kolmogorov (2006)	Tree-Reweighted Message Passing	Node-adjacent, anisotropic
SRMP	Kolmogorov (2015)	Sequential Reweighted Message Passing	
MPLP	Globerson and Jaakkola (2008)	Max-Product Linear Programming	Edges
MPLP++	Tourani et al. (2018)		
DMM	Shekhovtsov et al. (2016)	Dual Minorize-Maximize	Chains, hierarchical
TBCA	Sontag and Jaakkola (2009)	Tree Block Coordinate Ascent	Trees, sequential
	Tarlow et al. (2011)	Dynamic Tree Block Coordinate Ascent	Dynamic trees, sequential

Table 1: Surveyed block-coordinate ascent algorithms.

it is known that these duals are equivalent in the optimum (Wainwright et al., 2005; Savchynskyy, 2019), it has been believed that optimizing a stronger dual can be more efficient. Works (Meltzer et al., 2005; Ruozzi and Tatikonda, 2013) proposed a unified view of several MAP and sum-product algorithms as BCA methods. However, the dual objectives were different per method (derived from different region graphs (Meltzer et al., 2005), resp. splittings (Ruozzi and Tatikonda, 2013)) and the algorithms operate with messages and beliefs. We consider a single dual for all methods, following the more recent understanding of TRW-S (Kolmogorov, 2015), and all algorithms are explicitly updating the same dual variables.

We study the issue of non-uniqueness of the block maximizers in BCA methods and their influence on the overall algorithmic efficiency. Tourani et al. (2018) shows that MPLP method can be significantly improved by a small modification in the choice of block maximizers. We generalize these results to chain and tree subproblems. Werner and Průša (2019) study the effect on fixed points.

Our code is available at https://gitlab.com/tourani.siddharth/spam-code. Proofs of all mathematical statements can be found in the appendix.

2. MAP INFERENCE WITH BCA

MAP-Inference Problem Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with the *node* set $\mathcal{V}$ and *edge* set $\mathcal{E}$. A *labeling* $y\colon \mathcal{V} \to \mathcal{Y}$ assigns to each node $u \in \mathcal{V}$ a discrete *label* $y_u \in \mathcal{Y}$, where $\mathcal{Y}$ is some finite set of labels, w.l.o.g. assumed the same for all nodes. For brevity we will denote edges $\{u, v\} \in \mathcal{E}$ as just uv. For each node $u \in \mathcal{V}$ and edge $uv \in \mathcal{E}$ there are associated the following local cost functions: $\theta_u(s) \geq 0$ is the cost of a label $s \in \mathcal{Y}$ and $\theta_{uv}(s,t) \geq 0$ is the cost of a label pair $(s,t) \in \mathcal{Y}^2$, where the non-negativity is assumed w.l.o.g. Let also $\mathrm{Nb}(u)$ denote the set of neighbors of node u in $\mathcal{G}$.

In the well-known paradigm of MRF / CRF models, the posterior probability distribution is defined via the *energy* $E(y)$ as $p(y) \propto \exp(-E(y))$ and the *maximum a posteriori (MAP) inference* problem becomes equivalent to finding a labeling which minimizes the energy (total labeling cost): $y^* =$

$$\arg\min_{y \in \mathcal{Y}^{\mathcal{V}}} \Big[E(y \mid \theta) := \sum_{v \in \mathcal{V}} \theta_v(y_v) + \sum_{uv \in \mathcal{E}} \theta_{uv}(y_{uv}) \Big]. \quad (1)$$

Reparametrizations The representation of the energy function $E(y \mid \theta)$ as the sum of unary and pairwise costs is not unique: there exist many cost vectors θ' such that $E(y \mid \theta) = E(y \mid \theta')$ for all labelings $y \in \mathcal{Y}^{\mathcal{V}}$. Such cost vectors are called *equivalent*. All cost vectors equivalent to θ can be obtained as (*e.g.*, (Werner, 2007)):

$$\begin{aligned} \theta^{\phi}_u(s) &= \theta_u(s) - \textstyle\sum_{v \in \mathrm{Nb}(u)} \phi_{u,v}(s), \\ \theta^{\phi}_{uv}(s,t) &= \theta_{uv}(s,t) + \phi_{u,v}(s) + \phi_{v,u}(t) \end{aligned} \quad (2)$$

with some *reparametrization* vector $\phi = (\phi_{u,v}(s) \in \mathbb{R} \mid u \in \mathcal{V},\ v \in \mathrm{Nb}(u),\ s \in \mathcal{Y})$. This reparametrization is illustrated in Fig. 1(a). It is straightforward to see that when substituting (2) into (1) all contributions from ϕ cancel out and thus any reparametrized θ^{ϕ} is equivalent to θ (for the converse, that all equivalent costs do have such a representation see (Werner, 2007)).

Dual Problem The basic idea, pioneered in pattern recognition by (Schlesinger, 1976), is the following. In practice there exist oftentimes a reparametrization with the property that by selecting the label in each node independently as $y_u \in \arg\min \theta^{\phi}_u$ a good, or even optimal, solution is recovered. From an optimization perspective, this is captured by the lower bound: $D(\phi) :=$

$$\sum_{u \in \mathcal{V}} \min_{s \in \mathcal{Y}} \theta^{\phi}_u(s) + \sum_{uv \in \mathcal{E}} \min_{(s,t) \in \mathcal{Y}_{uv}} \theta^{\phi}_{uv}(s,t) \leq E(y^* \mid \theta), \quad (3)$$

obtained by applying the reparametrization in (1) and using the min-sum swap inequality. If there is a reparametrization such that the lower bound is tight and the minimizer $y_u \in \arg\min \theta^{\phi}_u$ in each node is unique, then y is the unique global optimum of (1). To tighten the lower bound we seek to maximize it in ϕ. It is known ((Werner, 2007; Savchynskyy, 2019)) that this maximization problem is dual to the natural linear programming relaxation of (1). The dual problem has the following advantages: (i) it is constraint-free; (ii)

(a) Dual Variables (b) Dual Blocks

Figure 1: (a) An edge block and its corresponding reparametrization components in the graphical notation of (Werner, 2007; Savchynskyy, 2019): nodes u, v are shown as grey ovals and circles representing possible labels. The lines connecting the labels represent label pairs (s, t) with associated pairwise costs $\theta_{uv}(s, t)$. For each set of pairwise costs connected to a particular label, there is one reparametrization coordinate $\phi_{u,v}(s)$ shown by a blue arc. (b) Different variable blocks. Highlighted are block sub-graphs and arcs indicating the variables considered. *Red:* node-adjacent block, *blue:* edge-block, *green* chain block.

it is composed of a sum of many simple concave terms, each of which is straightforward to optimize.

BCA algorithms Block-coordinate ascent methods exploit the structure of the dual by iteratively maximizing it w.r.t. different blocks of variables (subset of coordinates of ϕ) such that the block maximization can be solved exactly. Formally, let ϕ_F be the restriction of ϕ to a subset of its coordinates $F \subset \{(u, v, s) \mid u \in \mathcal{V}, v \in \text{Nb}(u), s \in \mathcal{Y}\}$, BCA algorithms perform the update:

$$\phi_F := \arg\max_{\phi_F} D(\phi) \tag{4}$$

with different blocks F in a static or dynamic order.

Constrained Dual For the purpose of this work, it is convenient to work with the constrained dual:

$$\max_{\phi} D(\phi) \quad \text{s.t.} \quad \theta^{\phi} \geq 0. \tag{5}$$

The equivalence can be shown by constructing for any solution ϕ to the unconstrained dual, a correction preserving the objective value and satisfying the constraints (Savchynskyy, 2019). We will formulate all BCA algorithms in this paper in a way that they maintain the feasibility to the constrained dual.

3. TAXONOMY OF BCA METHODS

We survey a number of BCA methods, listed in Table 1. Many of these methods are derived for different dual objectives and work with different sets of parameters. We reformulate them all as BCA methods on the dual (5) and identify the following important design components:

- Type of blocks used. This has a significant impact on algorithm efficiency. Larger blocks (such as chains or trees) lead to greater dual improvement, but optimizing over them requires more computations.
- Strategy of selecting which block to optimize at every step. A dynamic strategy may be more advantageous for some problems but has additional overhead costs.
- Type of the update applied. This is not systematically studied in the literature. The maximizer for each block is non-unique but instead it is any point in the optimal facet. Algorithms with drastically different behaviour are obtained, depending on the choice of the maximizer.

3.1 Choice of Variable Block

BCA algorithms (Table 1) exploit the following types of blocks that are tractable to be optimized over:

- **Node-adjacent blocks** $F_u = \{(u, v, s) \mid v \in \text{Nb}(u), \ s \in \mathcal{Y}\}$ consist of coordinates of the reparametrization vector that are "adjacent" to a node u, Fig. 1(b, red). These blocks are used in `TRW-S`, `MSD` and `CMP` algorithms.
- **Edge blocks** $F_{uv} = \{(u, v, s), (v, u, s) \mid s \in \mathcal{Y}\}$ containing all variables associated with an edge uv, see Fig. 1(b, blue). These are used in `MPLP` and `MPLP++` algorithms.
- **Chains and Trees** For a sub-graph $(\mathcal{V}', \mathcal{E}') \subset \mathcal{G}$ we select variables associated to all its edges: $F_{\mathcal{E}'} := \cup_{uv \in \mathcal{E}'} F_{uv}$, see Fig. 1(b, green). To optimize over such blocks, a dynamic programming subroutine is needed. Chain blocks are used *e.g.* in `DMM` (rows and columns of a grid graph). The `TRW-S` algorithm, which we introduced above as a node-adjacent BCA, simultaneously achieves optimality over a large collection of chains. Spanning trees are used in `TBCA` variants. We call edge, chain and tree blocks collectively as *subgraph blocks*.

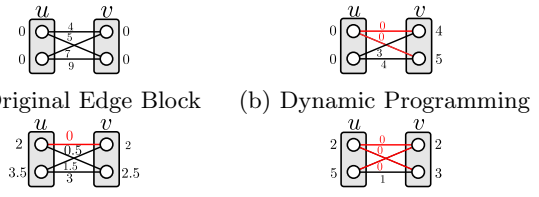
We will investigate which type of blocks and respective updates are more efficient.

3.2 Static vs. Dynamic Blocks

In dynamic `TBCA` (Tarlow et al., 2011) the trees are found dynamically by estimating where the dual can be increased the most (so-called local primal-dual gap (Tarlow et al., 2011)), which showed a significant practical speed-up in some applications (Tarlow et al., 2011). In other methods, the blocks are fixed in advance: *e.g.* rows and columns for grid graphs in `DMM`, single edge blocks in `MPLP`, spanning trees, selected greedily to cover the graph, in the static `TBCA`. We will investigate static and dynamic strategies for several update types.

3.3 Choice of The Local Maximizer

With the same blocks one could get very different algorithms depending on how the block maximizer is selected from the polyhedron of possible optimizers,



(a) Original Edge Block (b) Dynamic Programming

(c) MPLP (d) Handshake (HS)

Figure 2: Example of edge redistribution operations and the difference between non-maximal and maximal minorants; (a) initial edge block with some pairwise costs only; (b) DP update; (c) result of MPLP update; (d) result of Handshake update. Observe that node costs created by Handshake are strictly bigger than those of MPLP and more pairwise costs are made zero.

which we refer to as *update type*. We can systematize all used update types for node-adjacent blocks and subgraph-based blocks using several elementary operations. We now review them one by one.

Node-Adjacent Updates The update of blocks F_u works in two operations performing *aggregation* (6) and *distribution* (7) for every label s of u:

$$\forall v \in \mathrm{Nb}(u)\ \ \phi_{u,v}(s) := \phi_{u,v}(s) - \min_{l \in \mathcal{Y}} \theta^{\phi}_{uv}(s,l)\,, \quad (6)$$

$$\forall v \in \mathrm{Nb}(u)\ \ \phi_{u,v}(s) := \phi_{u,v}(s) + w_{u,v}\theta^{\phi}_{u}(s)\,, \quad (7)$$

where coefficients $w_{u,v}$ are non-negative and satisfy $\sum_{v\in\mathrm{Nb}(u)} w_{u,v} \leq 1$. After the aggregation, the reparametrized costs θ^{ϕ}_{uv} stay non-negative and label pairs that have zero cost are consistent with the minimizers of the unary reparametrized costs θ^{ϕ}_{u}. This step achieves block maximum (4). The purpose of the distribution step is to redistribute the cost excesses back to the edges of the block (leaving a fraction $1 - \sum_{v\in\mathrm{Nb}(u)} w_{u,v}$ at the node u) while preserving block optimality. In effect, the neighbouring nodes receive information about good labels for u. The MSD, CMP, dynamic programming (DP) and TRW-S algorithms are obtained by the respective setting of weights:

$$\texttt{MSD}\colon w_{u,v}{=}\tfrac{1}{|\,\mathrm{Nb}(u)|},\ \texttt{CMP}\colon w_{u,v}{=}\tfrac{1}{|\,\mathrm{Nb}(u)|+1}, \quad (8)$$

$$\texttt{DP}\colon w_{u,v}{=}[\![v{>}u]\!],\ \texttt{TRW-S}\colon w_{u,v}{=}\tfrac{[\![v>u]\!]}{\max\{N_{\mathrm{in}}(u),\,N_{\mathrm{out}}(u)\}},$$

where $[\![\cdot]\!]$ are Iverson brackets and the other details follow. The MSD and CMP algorithms do not express any preferences in direction (are isotropic) and the order of updating blocks is not as important. Updates of DP and TRW-S are anisotropic and depend on the order of the vertices. Let us see how DP updates work. Consider a chain graph and the chain ordering of nodes. For an inner node u there are two neighbouring nodes: $u-1$ and $u+1$. By choosing $w_{u,u+1} = 1$ and $w_{u,u-1} = 0$, we let all the excess costs be pushed forward and implement the forward pass of the Viterbi algorithm. TRW-S considers some order of processing of the nodes and applies coefficients $w_{u,v}$ such that for $v < u$ it is zero and for $v > u$ the coefficients are distributed evenly based on the numbers $N_{\mathrm{in}}(u)$, $N_{\mathrm{out}}(u)$ of incoming and outgoing edges in u w.r.t. the node order. Note that when there are more incoming edges than outgoing, these weights sum to less than one, *i.e.* some cost excess is left at the node u. It is clear that the choice of the block update and the order may be crucial in BCA methods.

In contrast to node-adjacent blocks, **subgraph blocks** overlap only in the nodes of the graph. Therefore, different redistribution strategies have been proposed in order to make the excess costs visible in all nodes of the processed block.

Edge Updates MPLP and MPLP++ methods consider edge blocks. MPLP performs the following symmetric update $\phi := \texttt{MPLP}_{u,v}(\theta,\phi)$:

$$\forall s{\in}\mathcal{Y},\quad \phi'_{u,v}(s) := \phi_{u,v}(s) + \theta^{\phi}_{u}(s), \quad (9a)$$

$$\forall t{\in}\mathcal{Y},\quad \phi'_{v,u}(t) := \phi_{v,u}(t) + \theta^{\phi}_{v}(t); \quad (9b)$$

$$\forall s{\in}\mathcal{Y},\ \phi_{u,v}(s) := \phi'_{u,v}(s) - \tfrac{1}{2}\min_{t\in\mathcal{Y}} \theta^{\phi'}_{uv}(s,t), \quad (10a)$$

$$\forall t{\in}\mathcal{Y},\ \phi_{v,u}(t) := \phi'_{v,u}(t) - \tfrac{1}{2}\min_{s\in\mathcal{Y}} \theta^{\phi'}_{uv}(s,t). \quad (10b)$$

The *aggregation* step (9) achieves that the costs in the nodes u, v become aggregated in the edge uv. Reparametrized costs $\theta^{\phi'}_{u}$, $\theta^{\phi'}_{v}$ become zero and $\theta^{\phi'}_{uv}(s,t) = \theta^{\phi}_{uv}(s,t) + \theta^{\phi}_{u}(s) + \theta^{\phi}_{v}(t)$ does not depend on the initial reparametrization components ϕ_{uv}, ϕ_{vu}. At this point the maximum over the edge block is found.

The *distribution* step (10a)-(10b) divides the aggregated cost in two halves and pushes the excesses from each half back to two nodes u, v, to make the preferred solution for the edge visible in the nodes. See Fig. 2c.

The Handshake (HS) update $\phi := \texttt{HS}_{u,v}(\theta,\phi)$ is used in MPLP++ and DMM. It differs in the distribution step. Let $\phi_{u,v}(s)$ be computed as in MPLP and $\phi_{v,u}(t)$ set to an arbitrary value. The Handshake update additionally performs:

$$\forall t \in \mathcal{Y},\quad \phi_{v,u}(t) := \phi_{v,u}(t) - \min_{s\in\mathcal{Y}} \theta^{\phi}_{uv}(s,t), \quad (11a)$$

$$\forall s \in \mathcal{Y},\quad \phi_{u,v}(s) := \phi_{u,v}(s) - \min_{t\in\mathcal{Y}} \theta^{\phi}_{uv}(s,t). \quad (11b)$$

This step pushes the still remaining cost excess from the edge to the nodes as illustrated in Fig. 2d. It leads to a strictly better improvement of the dual objective after the pass over all blocks and performs considerably better in experiments (Tourani et al., 2018). The step (11a) does not depend on the value of $\phi_{v,u}(t)$, which can be seen by moving $\phi_{v,u}(t)$ under the min and expanding the reparametrization. Therefore step (10b) may be omitted when computing HS.

Chain / Tree Updates The optimality over an edge, chain or a tree can be achieved by applying the following *dynamic programming* update $\phi := \texttt{DP}_{u,v}(\theta, \phi)$ (in the order of the chain or from leaves to the root of a tree):

$$\forall s \in \mathcal{Y}, \quad \phi_{u,v}(s) := \phi_{u,v}(s) + \theta^{\phi}_{u}(s), \tag{12a}$$

$$\forall t \in \mathcal{Y}, \quad \phi_{v,u}(t) := \phi_{v,u}(t) - \min_{s \in \mathcal{Y}} [\theta^{\phi}_{u,v}(s,t)]\,. \tag{12b}$$

The step (12a) aggregates the cost excess from node u to the edge uv and the step (12b) pushes the cost excess to note v. Observe that it can be written in the form of a node-adjacent update (6)-(7) by grouping the push step (12b) into v with the aggregation step (6) at v when processing the next edge vw.

TBCA algorithm uses DP to achieve optimality over a tree and then performs a pass in the reverse order, redistributing the costs with the following rDP update.

Redistribution DP update $\phi := \texttt{rDP}_{u,v}(\theta, \phi)$

$$\forall s \in \mathcal{Y}, \quad \phi_{u,v}(s) := \phi_{u,v}(s) + r\theta^{\phi}_{u}(s), \tag{13a}$$

$$\forall t \in \mathcal{Y}, \quad \phi_{v,u}(t) := \phi_{v,u}(t) - \min_{s \in \mathcal{Y}} [\theta^{\phi}_{u,v}(s,t)], \tag{13b}$$

where $0 \leq r \leq 1$ is a constant similar to the weights in the node-adjacent updates. The fraction r of cost excess is pushed forward to v and the fraction $1-r$ is left in the node u. TBCA detailed in Algorithm 1 and Fig. 3 redistributes cost excesses based on the size of the tree branch remaining ahead. TBCA was originally proposed for trees (Sontag and Jaakkola, 2009).

DMM works with chain subproblems and performs the redistribution hierarchically as explained in Algorithm 2 and Fig. 4. It was also originally proposed for the dual decomposition with chains (Shekhovtsov et al., 2016). One advantage of this method is that when the chain contains an edge with zero pairwise costs (no interactions), the processing becomes equivalent to redistribution in two chains independently. In contrast, the TBCA method would be confused in its estimate of the size of the subtree to push the excess to.

4. ANALYSIS

Subgraph-based blocks usually overlap over the nodes only (horizontal / vertical chains) or have a small overlap over the edges (spanning trees). Consider two blocks that overlap over nodes only. The representation of the information (costs of different solutions) which is available to one block about the other is limited to the reparametrized node costs θ^{ϕ}_{u} for all shared nodes u.

We identify this reparametrized unary potentials with modular minorants (Shekhovtsov et al., 2016), having clear analogies with minorants/majorants in pseudo-Boolean optimization (Boros and Hammer, 2002).

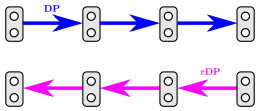


Figure 3: Tree-BCA computation. Blue arrows represent the DP update of the edge in that direction. Pink arrows represent the rDP operations. The TBCA update starts off by collecting costs at the end of the chain (right) and then redistributes with rDP with $r = (n-i)/n$ in the i'th backward step.

Algorithm 1 Tree-BCA Update on a Chain

Require: ϕ - starting reparametrization; $(\mathcal{V}', \mathcal{E}')$ - chain subgraph arranged as $\mathcal{V}' = \{1, \dots, n\}$ and $\mathcal{E}' = \{\{i, i+1\} \colon i = 1, \dots, n-1\}$.

1: **for** $i = 1, \dots, \mathrm{n}-1$ **do** ▷ Collects the costs at the chain end with dynamic programming
2: $\quad (\phi_{i,i+1}, \phi_{i+1,i}) := \texttt{DP}_{i,i+1}(\theta, \phi)$
3: **for** $i = n, \dots, 2$ **do** ▷ Distributes the costs in reverse order to the nodes
4: $\quad (\phi_{i-1,i}, \phi_{i,i-1}) := \texttt{rDP}_{i+1,i}(\theta, \phi)$ with $r = \frac{n-i}{n}$

4.1 Modular Minorants

A function $f \colon \mathcal{Y}^n \to \mathbb{R}$ of n discrete variables is called *modular*, if it can be represented as a sum of n functions of one variable: $f(y) = \sum_{i=1}^{n} f_i(y_i)$, $f_i \colon \mathcal{Y} \to \mathbb{R}$. The function $f(y) = \sum_{u \in \mathcal{V}'} \theta^{\phi}_{u}(y_u)$ is modular for any subset of nodes $\mathcal{V}' \subseteq \mathcal{V}$ and any reparametrization ϕ.

Definition 1. A modular function g is called a *(tight) minorant* of $f \colon \mathcal{Y}^n \to \mathbb{R}$, if (i) $g(y) \leq f(y)$ asnd (ii) $\min_{y \in \mathcal{Y}^n} f(y) = \min_{y \in \mathcal{Y}^n} g(y)$.

For the rest of this section we will assume that $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a subgraph defining a block of variables optimized at one step of a BCA algorithm and $E_{\mathcal{G}'}$ is the restriction of energy E to graph $\mathcal{G}'$ with the reparametrized costs θ^{ϕ}. A reparametrization ϕ is called *dual optimal on* $\mathcal{G}'$, if it is block-optimal in the sense of (4) w.r.t. block $F_{\mathcal{E}'}$. Minorants and dual optimal reparametrizations are closely related:

Theorem 1. Let $\mathcal{G}'$ be a tree and $\sum_{uv \in \mathcal{E}'} \min_{s,t} \theta^{\phi}_{uv}(s,t) = 0$. The function $g(y) = \sum_{u \in \mathcal{V}'} \theta^{\phi}_{u}(y_u)$ is a minorant for the energy $E_{\mathcal{G}'}(y)$ *if and only if* ϕ is dual optimal on $\mathcal{G}'$.

To put it differently, if $\mathcal{G}'$ defines a sub-graph block for a block-coordinate ascent method, then choosing amongst block optimal reparametrizations ϕ is equivalent, up to a constant, to choosing a modular minorant for the energy $E_{\mathcal{G}'}$.

Observe that, for a sub-graph block $(\mathcal{V}', \mathcal{E}')$, there are $2|\mathcal{E}'||\mathcal{Y}|$ reparametrization variables but only $|\mathcal{V}'||\mathcal{Y}|$

coordinates are needed to define a minorant. The minorant naturally captures the degrees of freedom that are important for subgraph-based BCA methods.

Minorants can be partially ordered with respect to how tightly they approximate the function. For two minorants g, g' we write $g' \geq g$ if $g'(y) \geq g(y)$ for all $y \in \mathcal{Y}^{\mathcal{V}'}$. Since our minorants are modular, the condition is equivalent to component-wise inequality $g'_u(s) \geq g_u(s)$ $\forall u \in \mathcal{V}', \forall s \in \mathcal{Y}$. The greater the minorant, the tighter it approximates the function. Hence, of interest are maximal minorants:

Definition 2 ((Shekhovtsov et al., 2016)). A minorant g is *maximal*, if there is no other minorant $g' \geq g$ such that $g'(y) > g(y)$ for some y.

For the best performance of a BCA method, it makes sense to select a maximal minorant and not just any minorant. To actually apply this idea to BCA methods, we show how the maximality property of a minorant translates back to reparametrizations:

Theorem 2. Let $\mathcal{G}'$ be a tree and reparametrization ϕ be dual optimal on $\mathcal{G}'$. The function $g(y) = \sum_{u \in \mathcal{V}'} \theta^{\phi}_u(y_u)$ is a maximal minorant if and only if $\forall uv \in \mathcal{E}'$ and $\forall s, t \in \mathcal{Y}$:

$$\min_{s' \in \mathcal{Y}} \theta^{\phi}_{uv}(s', t) = \min_{t' \in \mathcal{Y}} \theta^{\phi}_{uv}(s, t') = 0\,. \qquad (14)$$

With these results we can now draw conclusions about algorithms updating subgraph blocks. All BCA methods considered, as they achieve block optimality, construct minorants. However, many of them are not maximal. Minorants constructed by `MPLP` and `TBCA` are non-maximal. The change introduced in `MPLP++` achieves maximality as illustrated in Fig. 2d. This minor change brings more than an order of magnitude speed-up to the algorithm in some problem instances (Tourani et al., 2018). The correction can be extended to `TBCA`, also leading to improvements without any further changes, Sec. 5.2.

On the other side, the connection we established allows to interpret `DMM` as a BCA method working on the dual (5) and identify its reparametrization form as presented.

5. SYNTHESIS

Based on the above analysis and the experimental comparison of individual components of BCA methods, we synthesize the following BCA algorithm that appears to perform universally better in terms of achieved dual objective value versus time on a corpus of diverse problems. Here are the design choices that we made:

- We utilize chain blocks and the hierarchical minorant (`HM`, Algorithm 2 and Fig. 4) updates. These updates are the most expensive ones, but the maximality property and better redistribution of the excess costs pays off in practice.
- We observed that with the hierarchical minorant updates, selecting chain blocks dynamically does not give an improvement over a static set of chains, unlike in (Tarlow et al., 2011).
- We select chains automatically for a given graph by a new heuristic. This heuristic behaves favourably in both sparse regular graphs as well as dense graphs. This automatic choice allows the method to achieve a uniformly good performance over problems with different graph structure and connectivity.

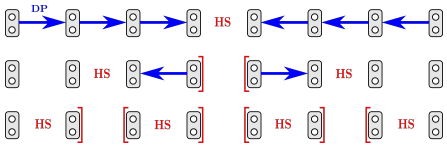


Figure 4: Hierarchical Minorant computation. The top level of hierarchy computes DP updates (blue arrows) towards the central edge. At the central edge the `Handshake` update (red HS) is applied. Then the same is applied recurrently to the two formed sub-chains. The messages that have been already computed are kept from preceding levels. When recurrence completes, the HS operation has been applied to every edge, resulting in a *maximal-minorant*.

Algorithm 2 Chain Hierarchical Minorant (`HM`) Update

Require: ϕ - starting reparametrization; chain subgraph $(\mathcal{V}', \mathcal{E}')$ arranged as $\mathcal{V} = \{1, \dots, n\}$ and $\mathcal{E} = \{\{i, i+1\} \colon i = 1, \dots, n-1\}$.

1: $i_L := \lfloor \frac{n}{2} \rfloor$; $i_R := i_L + 1$ ▷ Compute the left and right mid-points of the chain
2: **for** $i = 1, \dots, i_R - 1$ **do** ▷ Push costs from chain start to i_L
3: $(\phi_{i,i+1}, \phi_{i+1,i}) := \mathtt{DP}_{i,i+1}(\theta, \phi)$
4: **for** $i = n, \dots, i_R + 1$ **do** ▷ Push costs from the chain end to i_R
5: $(\phi_{i-1,i}, \phi_{i,i-1}) := \mathtt{DP}_{i,i-1}(\theta, \phi)$
6: $(\phi_{i_L,i_R}, \phi_{i_R,i_L}) := \mathtt{HS}_{i_L,i_R}(\theta, \phi)$ ▷ Handshake update on the middle edge
7: Call `HM` on two sub-chains $[1 \dots i_L]$ and $[i_R \dots n]$ if not empty

5.1 Experimental Setup

For a uniform evaluation over different problem types we formed three datasets grouping problems from different domains by their graph *connectivity*, which is the proportion of number of edges to the maximal possible number of edges $|\mathcal{V}||\mathcal{V}-1|/2$. These datasets are detailed in Fig. 9.

For objectiveness of comparison, we measure the com-

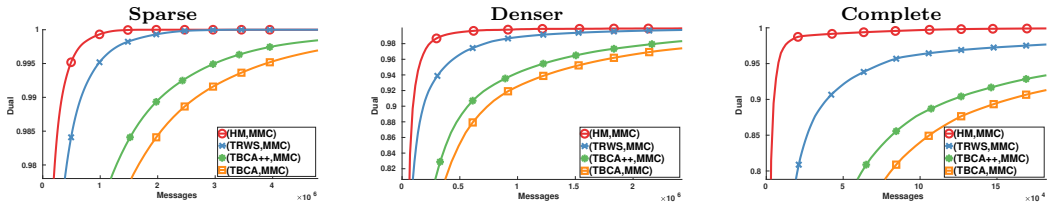


Figure 5: Comparison between TRW-S efficiently optimizing all monotonic chains with subgraph-based updates on a covering subset of monotonic chains. See description of datasets in Fig. 9.

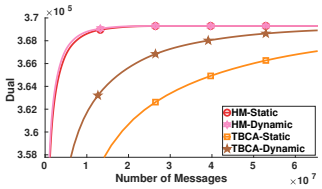


Figure 6: Comparison of static and dynamic spanning trees on tsukuba from the stereo dataset.

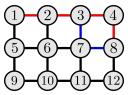


Figure 7: Strictly shortest paths. In this example the chain 1-2-3-4-8 and the chain 1-2-3-7-8 are both shortest paths to node 8 and therefore none of them is the strictly shortest path. The chain 1-2-3-4 is the unique (and hence strict) shortest path from 1 to 4.

putation cost in *messages*, the updates of the type $\min_t(a(t) + \theta_{uv}(s,t))$ that form the bulk of computation for all presented BCA methods. The number of messages is scaled by the ratio $\overline{|\mathcal{E}|}/|\mathcal{E}|$, where $|\mathcal{E}|$ is the number of edges in an instance and $\overline{|\mathcal{E}|}$ is the average over the dataset. These normalizations allow us to show average performance on the whole datasets.

5.2 TRWS vs. Subgraph Updates

TRW-S is selected as representing the most efficient node-adjacent update. In particular, it is much faster than CMP and MSD as shown *e.g.* in (Kolmogorov, 2015; Kappes et al., 2015). It was originally derived as a method for optimizing the dual decomposition of (1) with monotonic chains (Kolmogorov, 2006). We compared it to subgraph-based updates running on the same set of chains. Such direct comparison over datasets of different sparsity has not been conducted before. For a given graph we took a subset of maximum monotonic chains to cover all edges (exact details can be found in Appendix B). TRW-S is very efficient and takes $O(|\mathcal{E}|)$ messages to achieve optimality on all monotonic chains, including our covering subset. Two subgraph-based updates can be applied to optimize over chains in the covering subset sequentially: TBCA, taking $O(|\mathcal{E}|)$ messages as well and hierarchical minorant (HM, Algorithm 2) taking $O(|\mathcal{E}| \log |\mathcal{E}|)$ messages. The comparison in Fig. 5 on our broad corpus of problems shows that while TBCA is clearly inferior in performance to TRW-S, HM is actually performing significantly better than TRW-S. It works on the same subproblems as TBCA but the maximal minorant property justifies the extra computation time.

In this experiment we also evaluated an improved version of TBCA, denoted TBCA++, which modifies TBCA as follows: After each rDP update on uv the operation (11a) pushes the remaining cost back to v and thus achieves the maximal minorant conditions (14). This small change leads to a noticeable improvement, see Fig. 5. However, we can conclude that a better redistribution of cost excess done by HM is more important than the maximality alone.

5.3 Static vs. Dynamic

In the TBCA and HM methods, the choice of subproblems is not limited to monotonic chains. In (Tarlow et al., 2011) it was proposed to select spanning trees dynamically favouring node-edge pairs with the most disagreement as measured by the local primal-dual gap. We verified whether this strategy is beneficial with HM. Fig. 6 shows the comparison of dynamic spanning trees versus static spanning trees (a fixed collection selected greedily to cover all edges, see Appendix B). We reconfirm observations (Tarlow et al., 2011) on our corpus of problems that dynamic strategy is beneficial with TBCA updates. It does not however have a significant impact on the performance of HM updates. We therefore propose to use a static collection of subgraphs, optimized for a given graph.

5.4 Graph Adaptive Chain Selection

(Tourani et al., 2018) have shown that for densely connected graphs, edge-based updates are much faster than other methods. Since the MPLP++ update is equivalent to HM update on chains of length 1, this suggests that shorter chains are more beneficial in dense graphs. Intuitively, when there is a direct edge between nodes, the longer connections through other nodes become increasingly less important. On the contrary, in grid graphs MPLP++ is found inferior to TRW-S (Tourani et al., 2018) and the natural choice of row and column chains seems to be the best selection of sub-problems. Based on these observations there is a need for the sub-graphs

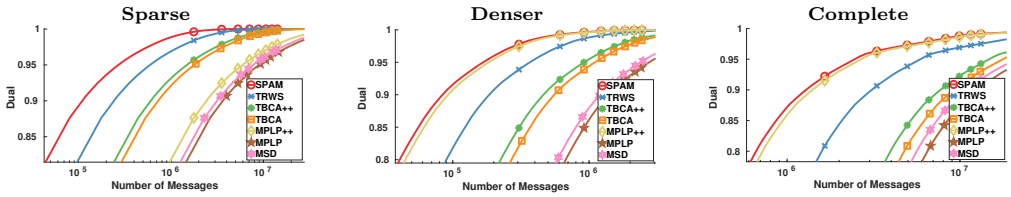


Figure 8: Algorithm comparison for sparse, denser and complete graphs following the experimental setup in Sec. 5.1. The minor difference between `MPLP++` and `SPAM` for complete graphs is explained by different order of computations. The corresponding runtime plots look qualitatively the same and are provided in the appendix.

Sparse: Problems on 4-connected grid graphs (less than 1% connectivity): instances of `stereo` (3 models with truncated linear pairwise costs and 16, 20 and 60 labels) and `mrf-inpainting` (2 models with truncated quadratic pairwise costs and 21838 and 65536 nodes) from the Middlebury MRF benchmark (Szeliski et al., 2008).

Denser: Problems with connectivity in between grids and complete graphs: `worms` (Kainmueller et al., 2014) (30 instances coming from the field of bio-imaging, 558 nodes with $20-65$ labels each, around 10% connectivity); `denser-stereo` (adds additional longer-range pairwise smoothness interactions to the `stereo` dataset, for each of 3 instances we create 4 denser variants of increased connectivity (20%, 30%, 40%, 50%)).

Complete: Problems with fully connected graphs (100% connectivity): `protein-folding` model (Yanover et al., 2008) instances of OpenGM benchmark (Kappes et al., 2015) (11 instances with $33-40$ nodes and up to 503 labels per node); `pose` 6D object pose estimation model (Michel et al., 2017) instances of (Tourani et al., 2018) (32 instances with 600-4800 variables and 13 labels).

Figure 9: Datasets of increasing connectivity used for benchmarking.

to be chosen adaptively to the graph topology. We use chain subproblems for their simplicity and better parallelization utility and propose the following informed heuristic:

- Select subproblems sequentially as shortest paths from the yet uncovered part of the graph;
- Chose *strictly* shortest paths, such that no other path of the same length connects the same nodes;
- Find the most distant pair of nodes connected by a strict shortest path.

An example of strict shortest path is given in Fig. 7. The algorithm implementing this heuristic, detailed in Appendix B, Algorithm 4, randomly picks a starting vertex, finds all strictly shortest paths from it (a variant of Dijkstra search), removes the longest traced shortest path from the graph and reiterates. This heuristic has the following properties: (i) in a complete graph, it selects edge subproblems; (ii) in a grid graph, irrespective of the input data ordering, it is likely to select large pieces of rows and columns (with some distortions due to greediness); (iii) in graphs with bottleneck long connections, these connections are likely to be covered with long chains.

5.5 The SPAM Algorithm

The synthesis of block selection via graph adaptive chain selection, as described in 5.4, and the hierarchical minorant updates we call the **S**hortest **P**ath **A**daptive **M**inorant (SPAM) algorithm. The adaptive chain selection has a linear complexity w.r.t. the size of the graph and takes only a fraction of a single iteration time of the main algorithm.

5.6 Final Experimental Evaluation

We tested the proposed `SPAM` algorithm against existing methods. Fig. 8 shows the summarized evaluation results. One can see that across all graph types `SPAM` consistently does well. It automatically adapts to the density of the graph, reducing to `MPLP++` for complete graphs, where `TRW-S` struggles. In grid graphs, where `TRW-S` uses the natural ordering, `SPAM` automatically finds sub-problems similar to rows and columns and achieves a significant improvement while `MPLP++` becomes inefficient. Detailed results per dataset and speed-up factors with confidence intervals are included in Appendix C.

6. CONCLUSION

We have reviewed, systematized and experimentally compared different variants of block-coordinate-ascent methods proposed to date. We have shown the updates for subgraph-based methods take the form of modular minorants and that maximal minorants outperform non-maximal ones. We experimentally compared existing methods as well as new combinations of basic components of BCA algorithms and synthesized a novel algorithm that is a synthesis of the best aspects of all methods. It additionally adopts block-size to the graph structure and delivers uniformly best performance across the tested datasets.

Acknowledgements This work was supported by the German Reserach Foundation ("Exact Relaxation-

Based Inference in GraphicalModels", DFG SA 2640/1-1) and the European Research Council (ERC European Unions Horizon 2020 research and innovation program, grant 647769). The computations were performed on an HPC Cluster at the Center for Information Services and High Performance Computing (ZIH) at TU Dresden. Alexander Shekhovtsov was supported by the project "International Mobility of Researchers MSCA-IF II at CTU in Prague" (*CZ*.02.2.69/0.0/0.0/18_070/0010457).

References

Alexander Shekhovtsov, Christian Reinbacher, Gottfried Graber, and Thomas Pock. Solving dense image matching in real-time using discrete-continuous optimization. In *CVWW*, page 13, 2016. ISBN 978-3-85125-388-7.

Gottfried Munda, Alexander Shekhovtsov, Patrick Knöbelreiter, and Thomas Pock. Scalable full flow with learned binary descriptors. In *GCPR*, pages 321–332, 2017. ISBN 978-3-319-66709-6.

Siddharth Tourani, Alexander Shekhovtsov, Carsten Rother, and Bogdan Savchynskyy. MPLP++: Fast, parallel dual block-coordinate ascent for dense graphical models. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Liang-Chieh Chen, Alexander Schwing, Alan Yuille, and Raquel Urtasun. Learning deep structured models. In *International Conference on Machine Learning*, pages 1785–1794, 2015.

Patrick Knöbelreiter, Christian Reinbacher, Alexander Shekhovtsov, and Thomas Pock. End-to-end training of hybrid CNN-CRF models for stereo. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Alexander Kolesnikov and Christoph H. Lampert. Seed, expand and constrain: Three principles for weakly-supervised image segmentation. In *ECCV*, pages 695–711, 2016.

Dmitrii Marin, Meng Tang, Ismail Ben Ayed, and Yuri Boykov. Beyond gradient descent for regularized segmentation losses. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Fabio Tosi, Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. Learning monocular depth estimation infusing traditional stereo knowledge. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Christian Payer, Michael Pienn, Zoltán Bálint, Alexander Shekhovtsov, Emina Talakic, Eszter Nagy, Andrea Olschewski, Horst Olschewski, and Martin Urschler. Automated integer programming based separation of arteries and veins from thoracic CT images. *Medical image analysis*, 2016.

Bogdan Savchynskyy, Jörg Hendrik Kappes, Paul Swoboda, and Christoph Schnörr. Global MAP-optimality by shrinking the combinatorial search area with convex relaxation. In *Advances in Neural Information Processing Systems*, pages 1950–1958, 2013.

Stefan Haller, Paul Swoboda, and Bogdan Savchynskyy. Exact MAP-inference by confining combinatorial search with LP relaxation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

MI Schlesinger and KV Antoniuk. Diffusion algorithms and structural recognition optimization problems. *Cybernetics and Systems Analysis*, 47(2):175–192, 2011.

Tamir Hazan and Amnon Shashua. Norm-product belief propagation: Primal-dual message-passing for approximate inference. *IEEE Transactions on Information Theory*, 56(12):6294–6316, 2010.

Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006.

Vladimir Kolmogorov. A new look at reweighted message passing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(5):919–930, 2015.

Amir Globerson and Tommi S. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in Neural Information Processing Systems*. 2008.

David Sontag and Tommi Jaakkola. Tree block coordinate descent for MAP in graphical models. In *Artificial Intelligence and Statistics*, pages 544–551, 2009.

Daniel Tarlow, Dhruv Batra, Pushmeet Kohli, and Vladimir Kolmogorov. Dynamic tree block coordinate ascent. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 113–120, 2011.

Jörg H. Kappes, Bjoern Andres, Fred A. Hamprecht, Christoph Schnörr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard X. Kausler, Thorben Kröger, Jan Lellmann, Nikos Komodakis, Bogdan Savchynskyy, and Carsten Rother. A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, pages 1–30, 2015.

Mengtian Li, Alexander Shekhovtsov, and Daniel Huber. Complexity of discrete energy minimization problems. In *European Conference on Computer Vision*, pages 834–852, 2016. ISBN 978-3-319-46475-6.

Tomas Werner. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7), 2007.

Bogdan Savchynskyy. Discrete graphical models — an

optimization perspective. *Foundations and Trends in Computer Graphics and Vision*, 11(3-4), 2019.

Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. MAP estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717, 2005.

Talya Meltzer, Chen Yanover, and Yair Weiss. Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 428–435. IEEE, 2005.

Nicholas Ruozzi and Sekhar Tatikonda. Message-passing algorithms: Reparameterizations and splittings. *IEEE Transactions on Information Theory*, 59(9):5860–5881, 2013.

Tomáš Werner and Daniel Průša. Relative interior rule in block-coordinate minimization. October 2019.

Michail I Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions. *Kibernetika*, 4(113-130):1, 1976.

E. Boros and P.L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 1-3(123):155–225, 2002.

Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. 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):1068–1080, 2008.

Dagmar Kainmueller, Florian Jug, Carsten Rother, and Gene Myers. Active graph matching for automatic joint segmentation and annotation of C. elegans. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 81–88. Springer, 2014.

Chen Yanover, Ora Schueler-Furman, and Yair Weiss. Minimizing and learning energy functions for side-chain prediction. *Journal of Computational Biology*, 15(7):899–911, 2008.

Frank Michel, Alexander Kirillov, Eric Brachmann, Alexander Krull, Stefan Gumhold, Bogdan Savchynskyy, and Carsten Rother. Global hypothesis generation for 6D object pose estimation. In *CVPR*, 2017.