An Efficient Branch-and-Bound Algorithm for Optimal Human Pose Estimation

Min Sun Murali Telaprolu Honglak Lee Silvio Savarese
Department of EECS, University of Michigan, Ann Arbor, MI 48109

Abstract

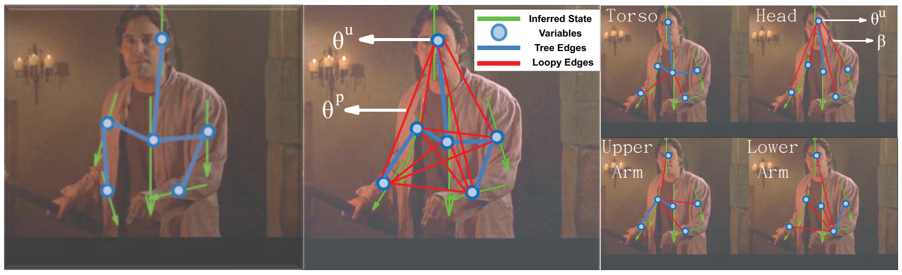
Human pose estimation in a static image is a challenging problem in computer vision in that body part configurations are often subject to severe deformations and occlusions. Moreover, efficient pose estimation is often a desirable requirement in many applications. The trade-off between accuracy and efficiency has been explored in a large number of approaches. On the one hand, models with simple representations (like tree or star models) can be efficiently applied in pose estimation problems. However, these models are often prone to body part misclassification errors. On the other hand, models with rich representations (i.e., loopy graphical models) are theoretically more robust, but their inference complexity may increase dramatically. In this work, we propose an efficient and exact inference algorithm based on branch-and-bound to solve the human pose estimation problem on loopy graphical models. We show that our method is empirically much faster (about 74 *times) than the state-of-the-art exact inference algorithm [21]. By extending a state-of-the-art tree model [16] to a loopy graphical model, we show that the estimation accuracy improves for most of the body parts (especially lower arms) on popular datasets such as Buffy [7] and Stickmen [5] datasets. Finally, our method can be used to exactly solve most of the inference problems on Stretchable Models [18] (which contains a few hundreds of variables) in just a few minutes.*

1. Introduction

Estimating the pose of humans (e.g., determining body part locations) from images and videos is a core problem in computer vision and it is critical in many applications such as human computer interaction, video surveillance and gaming. Because speed is an important requirement in most of these applications, researchers have focused on approaches that put a premium on efficiency. Among them, tree-structured models [6, 13, 5, 1, 15, 16, 22, 29] (Fig. 1a) are commonly used. A tree structure typically captures only the most informative spatial relationships (i.e., kinematic constraints) between pairs of parts since the location of one part is well constrained by the location of its connected parts (e.g., the hand location is constrained by the arm location). Inference in tree models can be done efficiently using dynamic programming. As a result, such models can strike a good balance between efficiency and estimation accuracy.

Despite their success, tree models are prone to some common misclassification errors. For example, left and right limbs are often misclassified because their appearance is typically very similar and their estimated locations tend to overlap in the image (over-counting evidence). To overcome these types of misclassification errors, more structured models such as the loopy graphical models (later referred as loopy model) have been proposed [9, 20, 31, 14, 26, 28] (Fig. 1b). By capturing interactions between a large number of pairs of parts, these methods are effective at improving pose estimation results at the expense of a significantly increased computational cost [21, 11]. For instance, methods based on cluster pursuit [21] become prohibitively slow when the number of states (i.e., number of part location hypothesis) is large since its time complexity is proportional to the number of states to the power of the cluster size (typically ≥ 3). Methods based on Branch-and-Bound (BB) [10] search are used in Bayesian networks with a large number of random variables [11], but they become extremely inefficient when the number of states becomes larger (as in the human pose estimation problem). This is because the search proceeds by instantiating each state of every random variable sequentially so that both time and memory usages increase dramatically when the number of states increases. To improve efficiency, i) greedy methods are used to reduce the part location hypothesis (e.g., selecting sparse interest points) [14, 31, 26] and/or ii) approximate inference approaches are applied [9, 31, 14, 26, 28].

In this work, we propose an efficient and exact inference algorithm based on BB to solve the human pose estimation problem on loopy models, where the number of part location hypotheses is large. Our BB algorithm is built upon our earlier BB algorithm [24] for solving MAP inference on general MRF. Our contribution is two-fold: i) similarly to linear programming relaxation, a novel bound is obtained by relaxing the loopy model into a mixture of star-models; ii) a special data structure (BMT) and an efficient search routine (OBMS) (see Sec. 4.2) are used to significantly reduce the time complexity for calculating the bound in each branch of the BB search. We empirically show that when



(a) Tree Model (b) loopy Model (c) Mixture of Star Model

Figure 1: Illustration of models. Panel (a,b,c) show graphical representations of the tree model, loopy model, and mixture of star models, respectively. Panel (c) enumerates star models with different parts (torso, head, upper-arm, and lower-arm) as the center parts. In all panels, every circle denotes a variable; every blue or red edge denotes the interaction between two variables in the tree or loopy model, respectively. The MAP assignments are shown in green arrows, where each arrow indicates the orientation of the body part.

the number of hypotheses per part is large, our new BB algorithm is an order of magnitude faster than state-of-the-art Cluster Pursuit (CP) method [21] in solving the exact MAP inference problem. Moreover, by extending a state-of-the-art tree model [16] to a loopy model, the estimation accuracy can be significantly improved (up to 5% for lower arm) on Buffy [7] and Stickmen [5] datasets. Finally, our method can exactly solve the MAP inference problem on the Stretchable Models [18] (which contains a few hundreds of variables) in just a few minutes, and achieves superior performance on a number of video sequences best represented by the pre-trained model (see Sec. 5.2).

In the following sections, we first describe the related work in Sec. 2. Then, we formulate the human pose estimation problem as the Maximum a Posteriori (MAP) inference problem over a Markov Random Field (MRF) in Sec. 3, and introduce our BB method and the efficient data structure in Sec. 4. Finally, we show experimental results in Sec. 5.

2. Related Work

Tree-models for human pose estimation have been introduced in [6] and extended to improve the robustness of part detectors [1, 13, 5] and the discriminative power of the pair-wise relations [15, 13, 29, 22]. Andriluka et al. [1] show that boosting classifiers can be used to detect parts very robustly, and the detections can be used by the tree models to improve the overall pose estimation accuracy. Sapp et al. [15] propose to use a pair-wise feature that depends on the image appearance (e.g., color, contour, segmentation, etc.) to enhance the discriminative power. Yang and Ramanan [29], and Sun and Savarese [22] use the concept of part-type (i.e., parts with specific orientation or foreshortening) to model pair-wise relations of a pair of part-types instead of a pair of parts. In this way, the pair-wise relations can capture co-occurrence of parts with specific orientation or foreshortening.

Loopy models have been successfully employed to solve human pose estimation problem. Interactions between many pairs of parts have been incorporated by [9, 26, 14] in order to encode information such as self-occlusion and color similarity of symmetric parts. As a result, the problem of over-counting evidence is mitigated. Wang et al. [28] and Zhu et al. [31] propose hierarchical models of parts across multiple scales such that parts at a lower level of the hierarchy are grouped into parts at a higher level of the hierarchy. In particular, Wang et al. [28] show that parts at a higher level of hierarchy might be easier to detect in isolation since they possess very distinctive appearance features (e.g., the whole human body is easier to detect than the hands).

A few works have been proposed to solve the MAP inference problem exactly for loopy models using efficient search algorithms. Tian and Sclaroff [25] propose an efficient BB algorithm for a tree model augmented with two additional pair-wise relations between left-right legs. The BB search is efficient since it only takes constant time to evaluate the bounds, this enabling the solution of problems with a large number of states. Notice, however, that the tightness of such bound guarantees efficient search only when the energy originated from the additional pair-wise relations is small (Fig.7 in [25]). This makes it hard for [25] to solve a model with many pair-wise interactions. Bergtholdt et al. [3] convert the inference problem over a fully connected model into a shortest path problem and propose an efficient A^* search method for solving it. The main drawback of the A^* search is that the branching factor of the search tree equals the number of states per variable (i.e., number of part location hypothesis). As a result, the method relies on a greedy procedure for pruning part hypotheses to ensure that the search problem is tractable. Cluster pursuit [21] is an alternative exact inference algorithm which searches for higher-order constraints to tighten the gap between approximated solution and optimal solution. However, since the time complexity of the algorithm is proportional to the number of part hypotheses to the power of the order of the constraints (i.e., number of variables involved in the constraints), the algorithm becomes prohibitively slow for problems with a large number of part hypotheses.

3. The Human Pose Estimation Problem

A loopy model is capable of capturing many pair-wise interactions of parts and can be modeled as pair-wise Markov Random Fields (MRFs). We define a MRFs model over a graph $G = \{\mathcal{N}, \mathcal{E}\}$ with a set of nodes (variables) $\mathcal{N}$ and a set of edges (pair-wise interactions) $\mathcal{E}$ as follows,

$$f(\mathbf{h}; \mathbf{\Theta}) = \sum_{i \in \mathcal{N}} \theta_i^u(h_i) + \sum_{ij \in \mathcal{E}} \theta_{ij}^p(h_i, h_j) \,, \qquad (1)$$

where $\mathbf{h} = (h_1, \ldots, h_{|\mathcal{N}|})$ is a set of part hypotheses h_i, $\mathbf{\Theta} = \{\theta_i^u; i \in \mathcal{N}\} \cup \{\theta_{ij}^p; ij \in \mathcal{E}\}$ is the set of unary potentials θ_i^u and pair-wise potentials θ_{ij}^p. The human pose estimation problem is equivalent to the Maximum a Posteriori (MAP) inference problem which finds the best assignment $\mathbf{h}^{MAP} \in \mathcal{H}_{\mathcal{N}}$ maximizing $f(\mathbf{h}; \mathbf{\Theta})$. $\mathcal{H}_{\mathcal{N}}$ denotes the joint hypotheses space $\mathcal{H}_1 \times \mathcal{H}_2 \cdots \times \mathcal{H}_{|\mathcal{N}|}$ and $h_i \in \mathcal{H}_i$, where $|\mathcal{N}|$ is the number of nodes in set $\mathcal{N}$.

Exact MAP inference over MRFs with large induced width is NP-hard [19]. Many approximate inference algorithms, such as loopy belief propagation [12] or generalized BP [30], have been proposed to obtain an efficient but non-optimal solution. Other approaches propose to relax the model into models which can be easily solved, such as star or tree models [27, 18]. We follow this intuition and relax the model into a mixture of star-models (Sec 3.1). This is useful since the MAP solution of a mixture of star-models can be obtained efficiently in time quadratically proportional to the number of states ($O(H^2)$). Notice, however, that the MAP solution of the relaxed model is unlikely to be the same as the MAP solution of the original loopy model. The key property that we prove in Sec 3.1 is that, when certain constraints (Eq. 3) are satisfied, the value of the MAP solution of the relaxed model is an upper bound of the value of the MAP solution of the original loopy model. This upper bound can be used by our newly proposed BB algorithm (Sec. 4) to find a MAP solution of the original loopy model. In Sec. 4, we describe the branching strategy of the newly proposed BB algorithm, and further introduce an efficient Branch-Max-Tree (BMT) data structure (Fig. 2) and an efficient Opportunistic Branch Max Search (OBMS) routine (Algorithm 5) to reduce the average time complexity of calculating the bound from $O(H^2)$ to $O(Q \log_2 H)$, where Q is typically smaller than the number of states H.

3.1. Mixture of Star-Models

A complex loopy MRF can be relaxed into a mixture of star-models $\{G_i; i \in \mathcal{N}\}$. We define G_i as a star-model consisting of node i and all its neighbours $\mathcal{N}_i$ according to the original graph G. In G_i, the only unary potential is $\theta_i^u(h_i)$ and the pair-wise potential on edge ji is $\beta_{ji}(h_j, h_i)$ (Fig. 1c). The relaxed model becomes

$$\begin{aligned} f^R(\mathbf{h},;\mathbf{\Theta}^u,\mathbf{B}) &= \sum_{i\in\mathcal{N}} \lambda_i(h_i;\mathcal{H}_\mathcal{N}) && (2)\\ &= \sum_{i\in\mathcal{N}}(\theta_i^u(h_i) + \sum_{j\in\mathcal{N}_i} \max_{\hat{h}_j\in\mathcal{H}_j} \beta_{ji}(\hat{h}_j,h_i))\ , \end{aligned}$$

where $\mathbf{B} = \{\beta_{ji}(h_j,h_i);\ (i,j) \in \mathcal{E}\}$ is the set of new pair-wise potentials, and $\lambda_i(h_i;\mathcal{H}_\mathcal{N})$ is the max-marginal value over $j \in \mathcal{N}_i$ in G_i. The advantage of the relaxed model is that the MAP inference is equivalent to the MAP inference over each star-model G_i (i.e., by calculating $h_i^* = \arg\max_{h_i\in\mathcal{H}_i} \lambda_i(h_i;\mathcal{H}_\mathcal{N})$ separately for all $i \in \mathcal{N}$), which can be done very efficiently using dynamic programming.

In order to relate the relaxed model to the original model, we further enforce the following constraints for $(i,j) \in \mathcal{E}, \forall h_i, h_j$,

$$\beta_{ji}(h_j,h_i) + \beta_{ij}(h_i,h_j) = \theta_{ij}^p(h_i,h_j)\ . \qquad (3)$$

Using Eq. 3, we show that the function value of the relaxed model (Eq. 2) can be expressed as an upper bound of the function value of the original model (Eq. 1) for any $\mathbf{h}$:

$$\begin{aligned} f^R(\mathbf{h};\mathbf{\Theta}^u,\mathbf{B}) &= \sum_{i\in\mathcal{N}}(\theta_i^u(h_i) + \sum_{j\in\mathcal{N}_i} \max_{\hat{h}_j\in\mathcal{H}_j} \beta_{ji}(\hat{h}_j,h_i))\\ &\geq \sum_{i\in\mathcal{N}}(\theta_i^u(h_i) + \sum_{j\in\mathcal{N}_i} \beta_{ji}(h_j,h_i))\\ &= f(\mathbf{h};\mathbf{\Theta})\ . && (4) \end{aligned}$$

Consequently, $f^R(\mathbf{h}^*;\mathbf{\Theta}^u,\mathbf{B})$ (for any $\mathbf{B}$ satisfying Eq. 3) is an upper bound of the function value of the MAP assignment $f(\mathbf{h}^{MAP};\mathbf{\Theta})$ since $f^R(\mathbf{h}^*;\mathbf{\Theta}^u,\mathbf{B}) \geq f^R(\mathbf{h}^{MAP};\mathbf{\Theta}^u,\mathbf{B}) \geq f(\mathbf{h}^{MAP};\mathbf{\Theta})$ where $\mathbf{h}^* = (h_1^*,\ldots,h_{|\mathcal{N}|}^*)$ is a MAP assignment of the mixture of star-models. Ideally, the gap between the upper bound and the value of the MAP assignment (i.e., $f^R(\mathbf{h}^*;\mathbf{\Theta}^u,\mathbf{B}) - f(\mathbf{h}^{MAP};\mathbf{\Theta})$) measures the tightness of the bound. However, such gap cannot be measured since the MAP assignment is unknown. Instead, we propose to measure the tightness of the bound by calculating the difference between the upper bound and the lower bound. The lower bound can be obtained in constant time as

$$\begin{aligned} LB(\mathbf{h}^*) &= f(\mathbf{h}^*;\mathbf{\Theta}) = \sum_{i\in\mathcal{N}} \theta_i^u(h_i^*) + \sum_{ij\in\mathcal{E}} \theta_{ij}^p(h_i^*,h_j^*)\\ &= \sum_{i\in\mathcal{N}}\left(\theta_i^u(h_i^*) + \sum_{j\in\mathcal{N}_i} \beta_{ji}(h_j^*,h_i^*)\right)\ , && (5) \end{aligned}$$

following the definition of $\mathbf{h}^{MAP}$ and Eq. 3.

There is a strong connection between our relaxed model and Linear Programming (LP) relaxation. Globerson and Jaakkola [8] derive a first-order Linear Programming (LP) relaxation, which intuitively can be interpreted as a mixture of star-models. This connection can be used to select $\mathbf{B}$ efficiently. For instance, $\mathbf{B}$ can be computed using the MP algorithm [8] (see more detail in our technical report [23]).

4. Inference

The MAP inference problem over a loopy model is hard since i) the hypothesis space $\mathcal{H}_\mathcal{N}$ is large, ii) typical methods, such as dynamic programming, which work well on tree-models, cannot be applied due to the complicated pair-wise relationships. We follow the intuition that many hypotheses are unlikely to be the MAP assignment. Hence, we propose a novel BB algorithm (Algorithm 1) which systematically searches for the MAP solution. At each step of the BB algorithm, the hypothesis space which is most likely to contain the MAP solution is branched into two subspaces, and the "likelihood" of each subspace is measured. Hypothesis subspaces are ranked according to the "likelihood" so that the hypothesis space which is most likely to contain the MAP solution is further branched during the next step. In such a way, the algorithm will avoid evaluating hypothesis spaces that are unlikely to contain the MAP solution. During the search, the upper bound of the value of the MAP assignment is used as the "likelihood" to guide

Algorithm 1 Our Proposed Branch and Bound algorithm

```
1:  Do Prep(True) (See Algorithm 2).
2:  Set $\mathcal{H}_\mathcal{N}$ as initial solution space and set a priority queue Q to empty.
3:  Do ($\mathbf{h}^*, UB$)=GetBound($\mathcal{H}_\mathcal{N}$) (Use Algorithm 6 to Efficiently evaluate Eq. 2).
4:  Set GLB = LB($\mathbf{h}^*$) (In Eq. 5).
5:  Insert ($\mathcal{H}_\mathcal{N}, UB, \mathbf{h}^*$) into Q.
6:  while true do
7:      ($\hat{\mathcal{H}}_\mathcal{N}, GUB, \mathbf{h}^*$) = pop(Q) (Get the branch with the global upper bound).
8:      if $\hat{\mathcal{H}}_\mathcal{N} \not\subset \mathcal{H}_\mathcal{N}$ then
9:          Do Prep(False) .
10:     end if
11:     Set $\mathcal{H}_\mathcal{N} = \hat{\mathcal{H}}_\mathcal{N}$.
12:     if $|GUB - GLB| \le \varepsilon$ then
13:         Return $\mathbf{h}^*$.
14:     else
15:         Do ($\mathcal{H}^1_\mathcal{N}, \mathcal{H}^2_\mathcal{N}$) = branching($\mathcal{H}_\mathcal{N}, \mathbf{h}^*$) (Branching Strategy (algo. 3)).
16:         Do ($\mathbf{h}^*_1, UB_1$)=GetBound($\mathcal{H}^1_\mathcal{N}$).
17:         Do ($\mathbf{h}^*_2, UB_2$)=GetBound($\mathcal{H}^2_\mathcal{N}$).
18:         GLB = max(LB($\mathbf{h}^*_1$),LB($\mathbf{h}^*_2$),GLB) (Get global LB).
19:         Insert ($\mathcal{H}^1_\mathcal{N}, UB_1, \mathbf{h}^*_1$) and ($\mathcal{H}^2_\mathcal{N}, UB_2, \mathbf{h}^*_2$) into Q.
20:     end if
21: end while
```

the search. Branch-and-bound [10] is guaranteed to reach the MAP solution when the most likely hypothesis space has zero gap between the upper and lower bound. Notice that the upper bound in Eq. 2 satisfies the zero gap requirement since, when the hypothesis space for each node i contains one single part hypothesis (i.e., $\mathcal{H}_i$ equals to $\{h_i\}$), $f^R(\mathbf{h}; \mathbf{\Theta}^u, \mathbf{B}) = f(\mathbf{h}, \mathbf{\Theta})$ according to Eq. 4. Therefore, in the worst case, the BB search will stop when the most likely hypothesis space contains only one hypothesis. We describe the branching strategy in Sec. 4.1 and introduce the efficient bound calculation algorithm (Algorithm 6) in Sec. 4.2.

4.1. Branching Strategy

At each step of the BB algorithm (line 15 in Algorithm 1), the hypothesis space which is most likely to contain the MAP solution is branched into two subspaces. The hypothesis space is split by splitting the hypothesis space of a selected variable. Notice that the hypothesis space of the variable is split geometrically since we order the hypotheses so that geometrically nearby part hypotheses are also nearby in the ordered list (line 3 in Algorithm 3). Inspired by Batra et al. [2] and similarly to [24], we use a scoring function that we call Node-wise Primal Dual Gap (NPDG) as a cue to select the variable for branching (line 2 in Algorithm 3). Notice that the upper bound in Eq. 2 is already the sum of node-wise upper bound $\lambda_i(h^*_i)$,

Algorithm 2 Preprocessing: $Prep(InitFlag)$

```
1:  for $i \in \mathcal{N}$ do
2:      set $BMT_i.Set(\{\lambda_i(h_i) : h_i \in \mathcal{H}_i\})$.
3:      if InitFlag then
4:          for $h_i \in \mathcal{H}_i$ do
5:              for $j \in \mathcal{N}_i$ do
6:                  Set $BMT_{ji}(h_i).Set(\{\beta_{ji}(h_j, h_i) : h_j \in \mathcal{H}_j\})$.
7:              end for
8:          end for
9:      end if
10: end for
```

Algorithm 3 Branching Strategy ($\mathcal{H}^1_\mathcal{N}, \mathcal{H}^2_\mathcal{N}$) =branching($\mathcal{H}_\mathcal{N}, \mathbf{h}^*$)

```
1: Input: $\mathcal{H}_\mathcal{N} = \langle \mathcal{H}_1 \times \cdots \times \mathcal{H}_N \rangle$; $\mathbf{h}^* = (h^*_1, \ldots, h^*_N)$.
2: Select $\mathcal{H}_{i^*}$ where $i^* = \arg\max_{i \in \mathcal{V}} \delta_i(h^*_i)$ (See Sec. 4.1).
3: Suppose $\mathcal{H}_{i^*} = [h_j \ldots h_k]$ (States are in a fixed order)
4: Set $\mathcal{H}^1_{i^*} = [h_j \ldots h_{\lfloor 0.5(j+k) \rfloor}]$; $\mathcal{H}^2_{i^*} = [h_{\lfloor 0.5(j+k) \rfloor + 1} \ldots h_k]$ (Split roughly in half).
5: Set $\mathcal{H}^1_\mathcal{N} = \langle \mathcal{H}_1 \times \cdots \times \mathcal{H}^1_{i^*} \cdots \times \mathcal{H}_N \rangle$; $\mathcal{H}^2_\mathcal{N} = \langle \mathcal{H}_1 \times \cdots \times \mathcal{H}^2_{i^*} \cdots \times \mathcal{H}_N \rangle$.
6: Output: $\mathcal{H}^1_\mathcal{N}$ and $\mathcal{H}^2_\mathcal{N}$.
```

where $h^*_i = \arg\max_{h_i \in \mathcal{H}_i}(\lambda_i(h_i))$. Similarly, the lower bound in Eq. 5 is also the sum of node-wise lower bound $\hat{\lambda}_i(\mathbf{h}^*) = \theta^u_i(h^*_i) + \sum_{j \in \mathcal{N}_j} \beta_{ji}(h^*_j, h^*_i)$. We define NPDG as $\delta_i(\mathbf{h}^*) = \lambda_i(h^*_i) - \hat{\lambda}_i(\mathbf{h}^*)$. Here, $\delta_i(h^*_i)$ is always non-negative since

$$\begin{aligned}\lambda_i(h^*_i) &= \theta^u_i(h^*_i) + \sum_{j \in N(i)} \max_{h_j \in \mathcal{H}_j} \beta_{ji}(h_j, h^*_i) \\ &\ge \theta^u_i(h^*_i) + \sum_{j \in N(i)} \beta_{ji}(h^*_j, h^*_i) = \hat{\lambda}_i(\mathbf{h}^*). \quad (6)\end{aligned}$$

Moreover, $\sum_{i \in \mathcal{V}} \delta_i(h^*_i) = 0$ implies that the exact solution is found, since by definition the sum of NPDG is the gap (i.e., $\sum_{i \in \mathcal{V}} \delta_i(h^*_i) = f^R(\mathbf{h}^*, ; \mathbf{\Theta}^u, \mathbf{B}) - f(\mathbf{h}^*, ; \mathbf{\Theta})$). These properties suggest that we should select the variable with the largest NPDG to greedily reduce the gap.

4.2. Efficient Bound

We observed that finding the maximum value over a branch of a 1D array is the most common operation while finding the MAP assignment in Eq. 2. In particular, this operation appears when computing:

- $\max_{\hat{h}_j \in \mathcal{H}_j} \beta_{ji}(\hat{h}_j, h_i)$ needs to be calculated for all pairs of $(j, i) \in \mathcal{E}$ and all hypotheses $h_i \in \mathcal{H}_i$. Hence, the overall time complexity is $O(EH^2)$, where E is the number of edges and H is the number of states (per node). Notice that every β is a constant value.

- $h^*_i = \arg\max_{h_i \in \mathcal{H}_i} \lambda_i(h_i; \mathcal{H}_\mathcal{N})$ needs to be calculated for all nodes $i \in \mathcal{N}$. Hence, the overall time complexity is $O(NH)$, where N is the number of nodes and H is the number of states (per node). Notice that $\lambda_i(h_i; \mathcal{H}_\mathcal{N})$ is a function of the hypothesis space ($\mathcal{H}_\mathcal{N}$) in the branch (i.e., not a constant value).

Since both computations will be repeatedly used in all branching steps, it is critical that they are implemented efficiently. In the following, we propose a data structure to efficiently find the maximum over a branch of a 1D array.

Branch-Max-Tree (BMT). The key idea of the BMT is to utilize a one-time preprocessing step to speed up the querying operation which is supposed to be repeated multiple times. Given an Array $A[1 \ldots H]$, a Branch-Max-Tree (BMT) is set up (denoted by BMT.Set(A) in Fig. 2) in order to efficiently answer queries of the form $\max_{k \in \mathcal{H}} A[k]$ (denoted by $BMT.\max(\mathcal{H})$ in Fig. 2). As illustrated in Fig. 2, all nodes keep the pointer to the maximum value of its children nodes in the BMT. The tree is set up in time and memory usage both linearly proportional to the size of the array. Once the tree is set up, the maximum value of a

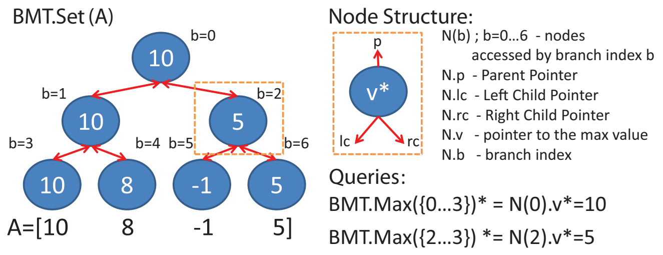

BMT.Set (A)
b=0
10
b=1
10
b=2
5
b=3
b=4
b=5
b=6
10
8
-1
5
A=[10 8 -1 5]
Node Structure:
p
v*
lc
rc
N(b) ; b=0...6 - nodes accessed by branch index b
N.p - Parent Pointer
N.lc - Left Child Pointer
N.rc - Right Child Pointer
N.v - pointer to the max value
N.b - branch index
Queries:
BMT.Max({0...3})* = N(0).v*=10
BMT.Max({2...3}) *= N(2).v*=5

Figure 2: Illustration of the Branch-Max-Tree (BMT). The left panel shows an example of a BMT set up from a simple Array A with only 4 elements. Notice that each node in the tree caches a pointer to the max value of its child nodes, and the max value is shown for illustration purposes. The top-right panel shows the data structure of a node used to construct BMT. The bottom-right panel shows that once BMT is built, each branch max query can be converted to a constant time look up from the corresponding node. Notice that the superscript $*$ denotes pointer dereferencing.

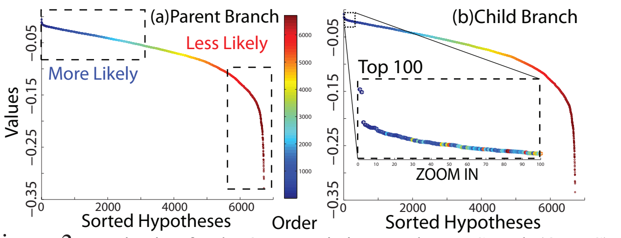


Figure 3: Motivation for the Opportunistic Branch Max Search (OBMS). Panel (a) shows the sorted $\lambda_i(h_i)$ value (y axis) for each hypothesis (x-axis) of the parent branch, where colors from blue to red represent the order from large to small values. Panel (b) shows the sorted $\lambda_i(h_i)$ value (y axis) for each hypothesis (x-axis) of the child branch, where the same color-code according to the order obtained in its parent branch is used. We clearly see that the top few hypotheses are mostly all blue. This implies that the top few hypotheses are very similar across the parent and child branches.

branch $\mathcal{H}$ can be simply looked up (in constant time) from a node in BMT, where all its succeeding leaf-nodes fully cover $\mathcal{H}$. The requirement of using BMT is that the values of the array A must be fixed.

Now we show the computation of $\max_{\hat{h}_j \in \mathcal{H}_j} \beta_{ji}(\hat{h}_j, h_i)$ can be sped up by using BMT. Since $\beta_{ji}(\hat{h}_j, h_i)$ for a specific h_i is a constant 1D array, $\max_{\hat{h}_j \in \mathcal{H}_j} \beta_{ji}(\hat{h}_j, h_i)$ can be obtained in $O(1)$ time, once the BMT (denoted by $BMT_{ji}(h_i)$) is set up at the beginning of the BB algorithm in $O(H)$ time. Hence, in line 1 of Algorithm 1, a set of pair-wise $BMTs$ (i.e.,$\{BMT_{ji}(h_i); h_i \in \mathcal{H}_i, (j,i) \in \mathcal{E}\}$) are set up in $O(EH^2)$ total time to speed up the query time computation, where E is the number of edges in the CRF. Notice that, in our earlier work [24], the computation is cast into a Range Maximum Query (RMQ) problem [4]. In this work, a simpler BMT data structure is used since the ranges (branches) are predefined according to the branching strategy in Algorithm 3. Most importantly, the second computation ($\arg\max_{h_i \in \mathcal{H}_i} \lambda_i(h_i; \mathcal{H}_\mathcal{N})$) cannot be handled by RMQ, but can be handled by BMT as described below.

When A has been changed, the BMT needs to be reset from scratch so that no speed-up is achieved. Hence, computing $\arg\max_{h_i \in \mathcal{H}_i} \lambda_i(h_i; \mathcal{H}_\mathcal{N})$ cannot be sped up by directly using the BMT. However, we observed that the value of $\lambda_i(h_i; \mathcal{H}_\mathcal{N})$ for different hypotheses are distributed in a large range (Fig. 3(a)). Most importantly, if we compare $\lambda_i(h_i; \mathcal{H}_\mathcal{N})$ in one branch with its child branch, we find that the maximal few hypotheses do not change much (Fig. 3(b)). This suggest that the maximal few hypotheses of one branch are likely to be the maximal few hypotheses of its child branch as well. Intuitively, we only need to find the maximum hypothesis among these few hypotheses. Therefore, we propose an Opportunistic Branch Max Search (OBMS) routine to speed up the computation.

Opportunistic Branch Max Search (OBMS). Instead of only knowing the Array $A[1 \dots H]$, now we assume that its element-wise upper bound $A^U[1 \dots H]$ (i.e., $A[h] \leq A^U[h];\ \forall h$) is also given. The opportunistic strategy to find $\max_{h \in \mathcal{H}} A[h]$ for any branch $\mathcal{H}$ is to test if the maximizer (i.e., $h^{U*} = \arg\max_{h \in \mathcal{H}} A^U[h]$) of A^U is also the maximizer of A. This can be done by checking whether the condition $A[h^{U*}] \geq A^U[h]$ is satisfied for all $h \in \mathcal{H} \setminus h^{U*}$. If the condition is not satisfied, we can update the upper bound $A^U[h^{U*}] = A[h^{U*}]$ and test the maximizer of the updated A^{U*} iteratively until the condition is satisfied. Using this opportunistic strategy, we avoid evaluating all elements in $\mathcal{H}$ which costs $O(|\mathcal{H}|)$.

In the OBMS routine, it is critical to obtain $h^{U*} = \arg\max_{h \in \mathcal{H}} A^U[h]$ and update $A^U[h]$ very efficiently. At the first glance, a priority queue seems to be a good data structure. This can be set up in $O(|\mathcal{H}|)$ time, but efficiently queried in $O(1)$ time and updated in $O(\log_2 |\mathcal{H}|)$ time. However, since we need to query for $h^{U*} = \arg\max_{h \in \mathcal{H}} A^U[h]$ multiple times in the BB search for different branches $\mathcal{H}$, a priority queue needs to be set up from scratch for each branch. Therefore, no speed-up is achieved.

Efficient BMT Update. We propose to set up a BMT for $A^U[h]$. A bottom-up procedure (Algorithm 4) efficiently updates the nodes in BMT along the path from the leaf-node corresponding to the updated element to the node corresponding to the branch (denoted by BMT.update$(\mathcal{H}, h^{U*}, A[h^{U*}])$). The time complexity is $O(\log_2 |\mathcal{H}|)$ (the same as the priority queue) since the update follows a single path in the BMT. The same BMT can be used for any query with branch $\hat{\mathcal{H}}$ which is the subset of $\mathcal{H}$. However, for other queries, the BMT needs to be reset from scratch with complexity $O(|\hat{\mathcal{H}}|)$ (line 9 in Algorithm 1).

Algorithm 4 Efficient Update Procedure for BMT: BMT.update($\mathcal{H}$,h,v)

```
1:  Input: $\mathcal{H}$ specifies the branch to be updated; $h$ specifies the leaf-node
    where the update starts; $v$ is the new updated value.
2:  Set $b_r = b(\mathcal{H})$ to be the branch index for the whole branch; $b_l =$
    $b(\{h\})$ to be the branch index of the leaf-node; the working node
    $N_w = N(b_l).p$ to be the parent of the leaf-node.
3:  Update $N(b_l).v^* = v$ .
4:  while $N_w.b \neq b_r$ do
5:      if $N_w.lc.v^* > N_w.rc.v^*$ then
6:          Update $N_w.v = N_w.lc.v$ .
7:      else
8:          Update $N_w.v = N_w.rc.v$ .
9:      end if
10:     Set $N_w = N_w.p$ .
11: end while
```

Algorithm 5 Opportunistic Branch Max Search:(h^*, v) =OBMS($\mathcal{H}_\mathcal{N}$,i)

```
1: Input: $\mathcal{H}_\mathcal{N}$ specifies the branch, $i$ specify the node index.
2: Set $h^*$=$NULL$ .
3: while true do
4:     Set $\hat{h} = BMT_i.\max(\mathcal{H}_i)$ .
5:     if $h^* \neq \hat{h}$ then
6:         Set $v = \theta^u_i(\hat{h})$ .
7:         for $j \in \mathcal{N}_i$ do
8:             Set $v = v + BMT_{ji}(\hat{h}).\max(\mathcal{H}_j)$ .
9:         end for
10:        Set $BMT_i.update(\mathcal{H}_i, \hat{h}, v)$ .
11:        $h^* = \hat{h}$ .
12:    else
13:        break.
14:    end if
15: end while
16: Return $v$.
```

Now we show that computation of $\arg\max_{h_i \in \mathcal{H}_i} \lambda_i(h_i; \mathcal{H}_\mathcal{N})$ can be sped up by using OBMS. By careful inspection, we found that $\lambda_i(h_i; \mathcal{H}_\mathcal{N})$ is the upper bound of $\lambda_i(h_i; \hat{\mathcal{H}}_\mathcal{N})$ when $\hat{\mathcal{H}}_\mathcal{N}$ is a subset of $\mathcal{H}_\mathcal{N}$. This is because

$$\max_{h_j \in \hat{\mathcal{H}}_j} \beta_{ji}(h_j, h_i) \leq \max_{h_j \in \mathcal{H}_j} \beta_{ji}(h_j, h_i);\ \hat{\mathcal{H}}_j \subset \mathcal{H}_j\ . \quad (7)$$

is true for all h_i and $(j, i) \in \mathcal{E}$. Given such a property, we set up a BMT_i for $\lambda_i(h_i; \hat{\mathcal{H}}_\mathcal{N})$ (in the pre-processing step) and use OBMS to efficiently calculate $\arg\max_{h_i \in \hat{\mathcal{H}}_i} \lambda_i(h_i; \hat{\mathcal{H}}_\mathcal{N})$ for all $i \in \mathcal{N}$. The routine takes $O(NQ \log_2 \hat{\mathcal{H}}_i) \leq O(NQ \log_2 H)$ instead of $O(N\hat{\mathcal{H}}_i) \leq O(NH)$, where Q is the number of trials in the OBMS and N is the number of nodes in the CRF. Typically $Q << H$ since we observed that the order of the top few hypotheses are not changing much (Fig. 3(b)). The OBMS routine is shown in Algorithm 5.

In summary, we propose to pre-process the data structure before BB search in $O(EH^2 + NH)$ time to reduce the time to calculate the bound from $O(EH^2)$ to $O(NQ \log_2 H)$ when a sub-branch is explored and $O(NH)$ otherwise. Notice that the overall time complexity of the algorithm also depends on the number of BB iterations B. Hence, the overall average time complexity becomes $O(BNH)$ (when only BMT is used) and $O(B_1 NQ \log_2 H + B_2 NH) < O(BNH)$ (when both BMT and OBMS are used), where B_2 is the number of times BMT needs to be re-initialized (line 9 of Algorithm 1) and $B_1 + B_2 = B$. The time complexity of Cluster Pursuit (CP) method [21] is $O(CPH^q)$, where P is the number of message passing iterations, q is the size of the clusters pursued, and C is the number of clusters with size q. Therefore, our BB algorithm is faster than CP when $BN < CPH^{q-1}$ is satisfied. In our experiment,

Algorithm 6 Get Bounds: $(\mathbf{h}^*, UB)$=GetBound($\mathcal{H}_\mathcal{N}$)

```
1: set $UB = 0$.
2: Definte $\mathbf{h}^* = (h^*_1, \ldots, h^*_{|\mathcal{N}|})$.
3: for $i \in \mathcal{N}$ do
4:     Get $(h^*_i, v_i)$ =OBMS($\mathcal{H}_\mathcal{N}$,$i$).
5:     Set $UB = UB + v_i$.
6: end for
7: Return $(\mathbf{h}^*, UB)$.
```

	Buffy				Stickmen	
Parts	Ours F	Ours 13	Ours 7	CPS	Ours F	CPS
Head	**99.15**	**99.15**	**99.15**	**99.15**	**99.44**	99.17
Torso	**99.57**	**99.57**	**99.57**	**99.57**	**99.72**	**99.72**
RUA	**95.30**	93.59	93.16	**95.30**	**82.50**	82.22
LUA	**92.31**	**92.31**	**92.31**	91.88	80.28	**81.67**
RLA	**63.25**	59.83	60.26	59.83	**56.94**	54.44
LLA	**64.53**	62.39	61.97	59.83	**53.89**	51.94

Table 1: Pose estimation accuracy of different variants of our models compared to CPS on Buffy and PASCAL Stickmen datasets. Ours F, Ours 13, and Ours 7 denote our fully connected model, the model with 13 pair-wise relationships (full model excluding 2 relationships of symmetric arm pairs), and the model with 7 pair-wise relationships (tree model with 2 additional symmetric arm relationships), respectively.

we empirically demonstrate that our method is faster than CP method when H is large (a few hundred), which implies $BN < CPH^{q-1}$.

5. Experiments

We evaluated performances and computational efficiency of our algorithm applied on loopy models and compare it against: i) existing tree-based models; 2) state-of-the-art inference algorithms. The first comparison is against the state-of-the-art tree model introduced by [16]. In order to guarantee a fair comparison with [16] we extend the cascade pictorial structure (CPS) [16] into a loopy model by capturing pair-wise part relationships other than the kinematic constraints. Notice that the same features, types of classifiers, and learning procedures are used to build and train the loopy model. We show that: i) the loopy model achieves better accuracy than the baseline tree model, ii) our proposed BB approach on the loopy model is much faster (74 times) than another state-of-the-art exact inference algorithm [21]. Furthermore, we show our novel BB algorithm can efficiently and exactly solve problems even with hundreds of variables. This analysis was done by using the Stretchable Models (SM) [18] to estimate human body joint locations across multiple frames (up to 30 frames). We conducted all the experiments on a 64-bit 16-Core Intel(R) Xeon(R) 2.40GHz CPU with 48GB RAM, the algorithms are implemented in single thread C++, and the time reported is in cpu-time (via the c++ clock() function).

5.1. Extended CPS Model

CPS is an upper body tree model with 6 articulated parts (i.e., head, torso, left/right-upper-arms, and left/right-lower-arms) which are parametrized by the location (x, y) and orientation (μ) of the part (i.e., $h = (x, y, \mu)$). The model achieves impressive performances by capturing more sophisticated pair-wise relationships than just geometric relationships using segmentation, contour, shape, and color features. In the CPS model, 5 decision-tree-based classifiers are trained to predict the strength of pair-wise relationships given the features. On top of the existing 5 classifiers, we further train 10 additional decision-tree-based classifiers and extend the model into a fully connected pair-wise model (i.e., a loopy model). Since now all the classifiers are trained independently, we treat the responses of the classifiers as the features $\mathbf{\Psi}$ and assume all potentials are linearly related to a set of model parameters such that the overall model is

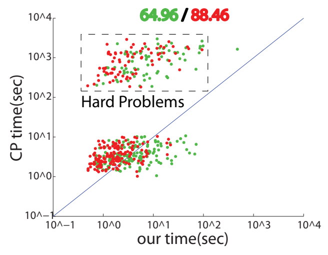


Figure 4: Scatter plot for the time comparison between the CP method (y axis) and our methods (x axis) on the Buffy dataset. Green indicates results of our approach without OBMS and red indicates results of our full BB approach. The two percentages on top of figure indicate how many times our two approaches are faster than the CP respectively.

linearly related to the parameters as:

$$f(\mathbf{h};\mathbf{w},I) = \sum_{i\in\mathcal{N}} w_i^T \psi_i(h_i,I) + \sum_{ij\in\mathcal{E}} w_{ij}^T \psi_{ij}(h_i,h_j,I)\ , \qquad (8)$$

where $\mathbf{w} = \{w_i,\ldots,w_{ij},\ldots\}$ is the set of all model parameters, $\psi_i(h_i,I)$ and $\psi_{ij}(h_i,h_j,I)$ are the unary and pair-wise features, respectively, and I is the image information. For conciseness, we define $f(\mathbf{h};\mathbf{w},I) = \mathbf{w}^T\mathbf{\Psi}(\mathbf{h},I)$, where $\mathbf{w}$, $\mathbf{\Psi}(.)$, and $\mathbf{h}$ are in the concatenated vector forms. The model is learned using the max-margin formulation (see technical report [23]).

We conduct experiment on both Buffy [7] and PASCAL Stickmen [5] dataset following the same experiment setup in [16]. The pose estimation performance is shown in Percentage of Correct Parts (PCP) for each part in Table 1. We report the CPS performance reproduced by the public available code released by Sapp et al. [16]. We also explore the effect of the connectivity of the model by training two sub-models with 13 and 7 pair-wise interactions on the Buffy dataset. Our fully connected model ("Ours F" in Table 1) outperforms the sub-models and CPS for most parts on both datasets. Moreover, our method achieves an average PCP of 85.7% which is significantly better than another fully-connected model [26] (67.6%) and on par with the state-of-the-art method [29] (89.1%) on the Buffy dataset. We also compare the time efficiency of different variants of our methods against a state-of-the-art Cluster Pursuit (CP) exact inference method. Notice that we calculate the bound in Eq. 2 by setting $\beta = 0.5\theta$ in this experiment, since it is more costly to do message passing to search for β at the beginning. Our method takes 0.28 hours in total to recognize poses in the whole Buffy dataset which contains 249 testing images. This is 74 times faster than CP method (20.83 hours). A scatter plot in Fig. 4 shows the time comparison for each example. It shows that our method with OBMS (red dots in Fig. 4) is faster than our method without OBMS (green dots in Fig. 4) (on average 2.5 times faster). Moreover, we identify two groups of examples. The group on the top is a set of hard examples since the CP method is required to search for more complex constraints in order to solve these problems. We observe that our method is faster than CP in 88% of the images in the dataset. Moreover, our BB algorithm requires less memory usage (on average 640MB) than the CP method (on average 7GB).

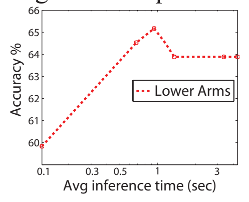


Figure 5: Trade-off between accuracy (y axis in PCP) and efficiency (x axis in time) of lower arms.

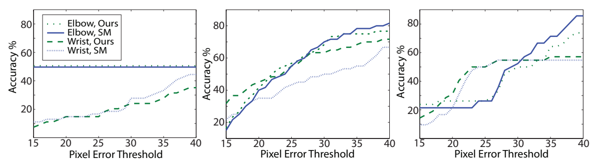


Figure 6: Quantitative results on three sequences in the VideoPose2.0 testset that are best represented by the pre-trained model. The predicted joint location is correct if its distance between the ground truth location is smaller than the specified pixel error threshold (x-axis). In the first column, both methods (our method and Stretchable Models (SM)) only detect half of the elbows. In the second sequence (Center), our method achieves almost consistently $\sim$ 10% better wrist accuracy than SM does. In the last sequence (Right), our method obtains better accuracy when the pixel error threshold is small for both elbow and wrist.

We also explore the trade-off between the pose estimation accuracy and inference time by allowing our method to stop early (increasing ϵ in line 12 of Algorithm 1). As shown in Fig. 5, when we allow approximate inference to run on average for 1.5 sec, the algorithm already reaches the same performance as the exact inference algorithm which takes 4.5 sec on average. Typical results of both the loopy model and the original tree model [16] on both Buffy and Stickmen datasets are shown in Fig. 7.

5.2. Stretchable Model (SM)

Sapp et al. [18] propose the Stretchable Model (SM) which models 6 body joint locations in each frame and captures interactions within frames as well as across consecutive frames. Since no existing methods can solve exact inference efficiently on such a large loopy model ($\sim$200 variables and a few hundred states per variable), they propose multiple inference techniques to solve the joint estimation problem: A) exact inference on relaxed models, B) approximate inference on a full model (dual decomposition). Their experimental results on VideoPose2.0 dataset [17] show that (A) is both more efficient and accurate than (B). In this experiment, we first use message passing to select the best β in order to avoid having much looser upper bound. We show that our BB algorithm can be directly applied to exactly infer the MAP solution over their pre-trained model. 13 out of 18 test sequences are solved within 20 minutes (on average 5.546 minutes). On the other hand, a dual decomposition approximate inference algorithm [8] can only solve 4 out of 18 problems. Moreover, CP method can only solve the same 4 problems even within one hour. Interestingly, although our method solves MAP estimation exactly, our method achieves similar elbow prediction accuracy but $\sim$ 5% lower wrist prediction accuracy. Our result and Sapp et al.'s conclusion suggest that the pre-trained model is not representing the video sequences well. Indeed, we discover that the learned model typically assigns much lower values to the ground truth assignments than it does with the values of the MAP assignments (on average 60% smaller). This means that the model often does not agree with the ground truth assignments. For example, in the first test sequence, the values of the MAP, Sapp et al.'s approximate inference, and the ground truth assignments are 20755, 17901, 9257,

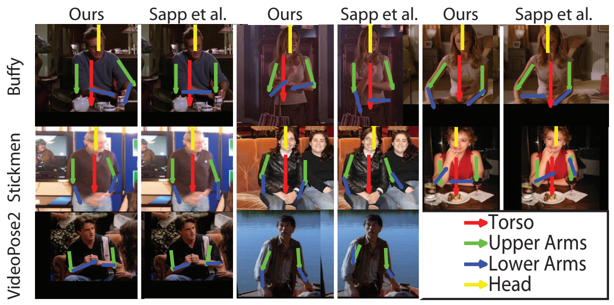


Figure 7: Typical results from Buffy, Pascal Stickmen, and VideoPose2 datasets shown in Stickmen representation from top to bottom, respectively. In each set of results, we show our result on the left and the Sapp et al.'s result on the right.

respectively. The value of the ground truth assignment is closer to the value of Sapp et al.'s approximate inference assignment than to the value of the MAP assignment. We follow this observation and select the top 3 sequences where the value of the ground truth assignment is closer to value of the MAP assignment with respect to the absolute difference between the values of the ground truth and Sapp et al.'s approximate inference assignments. In these cases, exact inference obtained by our method achieves comparable or superior accuracy (Fig. 6). Typical estimated body joint locations are shown in Fig. 7. This suggests that a better set of model parameters must be learned to fully demonstrate the power of the loopy model.

6. Conclusion

We have shown that our efficient and exact inference algorithm is 74 times faster than the state-of-the-art exact inference algorithm [21]. This enables the possibility of learning and applying loopy models to solve the pose estimation problem from a single image. We have shown that this does yield superior results in estimating body parts (e.g., 5% improvement for lower arm over a state-of-the-art method). We further show that our algorithm is general enough to solve problems with both a large number of variables ($\sim$ 200) and hundreds of states per variable in just a few minutes. From the results of the stretchable model experiment, we believe that learning parameters of complex models to achieve accurate performance while maintaining inference efficiency is an interesting future research direction.

Acknowledgements

We acknowledge the support of the ONR grant N000141110389, ARO grant W911NF-09-1-0310, and the Google Faculty Research Award.

References

[1] M. Andriluka, S. Roth, and B. Schiele. Pictorial structures revisited: People detection and articulated pose estimation. In *CVPR*, 2009. 1, 2

[2] D. Batra, S. Nowozin, and P. Kohli. Tighter relaxations for MAP-MRF inference: A local primal-dual gap based separation algorithm. In *AISTATS*, 2011. 4

[3] M. Bergtholdt, J. Kappes, S. Schmidt, and C. Schnrr. A study of parts-based object class detection using complete graphs. *IJCV*, 2009. 2

[4] O. Berkman and U. Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 1993. 5

[5] M. Eichner and V. Ferrari. Better appearance models for pictorial structures. In *BMVC*, 2009. 1, 2, 7

[6] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 2005. 1, 2

[7] V. Ferrari, M. M. Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. In *CVPR*, 2008. 1, 2, 7

[8] A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *NIPS*, 2008. 3, 7

[9] H. Jiang and D. R. Martin. Global pose estimation using non-tree models. In *CVPR*, 2008. 1, 2

[10] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 1960. 1, 4

[11] R. Marinescu and R. Dechter. Best-first AND/OR search for graphical models. In *AAAI*, 2007. 1

[12] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988. 3

[13] D. Ramanan. Learning to parse images of articulated bodies. In *NIPS*, 2006. 1, 2

[14] X. Ren, A. C. Berg, and J. Malik. Recovering human body configurations using pairwise constraints between parts. In *ICCV*, 2005. 1, 2

[15] B. Sapp, C. Jordan, and B. Taskar. Adaptive pose priors for pictorial structures. In *CVPR*, 2010. 1, 2

[16] B. Sapp, A. Toshev, and B. Taskar. Cascaded models for articulated pose estimation. In *ECCV*, 2010. 1, 2, 6, 7

[17] B. Sapp, D. Weiss, and B. Taskar. Sidestepping intractable inference with structured ensemble cascades. In *NIPS*, 2010. 7

[18] B. Sapp, D. Weiss, and B. Taskar. Parsing human motion with stretchable models. In *CVPR*, 2011. 1, 2, 3, 6, 7

[19] S. E. Shimony. Finding maps for belief networks is NP-hard. *Artificial Intelligence*, 2008. 3

[20] L. Sigal and M. J. Black. Measure locally, reason globally: Occlusion-sensitive articulated pose estimation. In *CVPR*, 2006. 1

[21] D. Sontag, T. Meltzer, A. Globerson, Y. Weiss, and T. Jaakkola. Tightening LP relaxations for MAP using message-passing. In *UAI*, 2008. 1, 2, 6, 8

[22] M. Sun and S. Savarese. Articulated part-based model for joint object detection and pose estimation. In *ICCV*, 2011. 1, 2

[23] M. Sun, M. Telaprolu, H. Lee, and S. Savarese. An efficient branch-and-bound algorithm for optimal human pose estimation. Technical report. http://www.eecs.umich.edu/ sunmin/. 3, 7

[24] M. Sun, M. Telaprolu, H. Lee, and S. Savarese. Efficient and exact MAP-MRF inference using branch and bound. In *AISTATS*, 2012. 1, 4, 5

[25] T.-P. Tian and S. Sclaroff. Fast globally optimal 2D human detection with loopy graph models. In *CVPR*, 2010. 2

[26] D. Tran and D. Forsyth. Improved human parsing with a full relational model. In *ECCV*, 2010. 1, 2, 7

[27] Y. Wang and G. Mori. Multiple tree models for occlusion and spatial constraints in human pose estimation. In *ECCV*, 2008. 3

[28] Y. Wang, D. Tran, and Z. Liao. Learning hierarchical poselets for human parsing. In *CVPR*, 2011. 1, 2

[29] Y. Yang and D. Ramanan. Articulated pose estimation using flexible mixtures of parts. In *CVPR*, 2011. 1, 2, 7

[30] J. S. Yedidia, W. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. Technical report, Mitsubishi Electrical Research Laboratories, 2002. 3

[31] L. L. Zhu, Y. Chen, Y. Lu, C. Lin, and A. Yuille. Max margin AND/OR graph learning for parsing the human body. In *CVPR*, 2008. 1, 2