Can We Learn Heuristics For Graphical Model Inference Using Reinforcement Learning?

Safa Messaoud, Maghav Kumar, Alexander G. Schwing
University of Illinois at Urbana-Champaign
{messaou2, mkumar10, aschwing}@illinois.edu

Abstract

Combinatorial optimization is frequently used in computer vision. For instance, in applications like semantic segmentation, human pose estimation and action recognition, programs are formulated for solving inference in Conditional Random Fields (CRFs) to produce a structured output that is consistent with visual features of the image. However, solving inference in CRFs is in general intractable, and approximation methods are computationally demanding and limited to unary, pairwise and hand-crafted forms of higher order potentials. In this paper, we show that we can learn program heuristics, i.e., policies, for solving inference in higher order CRFs for the task of semantic segmentation, using reinforcement learning. Our method solves inference tasks efficiently without imposing any constraints on the form of the potentials. We show compelling results on the Pascal VOC and MOTS datasets.

1. Introduction

Graphical model inference is an important combinatorial optimization task for robotics and autonomous systems. Despite significant progress in recent years due to increasingly accurate deep net models, challenges such as inconsistent bounding box detection, segmentation or image classification remain. Those inconsistencies can be addressed with Conditional Random Fields (CRFs), albeit requiring to solve an inference task which is of combinatorial complexity.

Classical algorithms to address combinatorial problems come in three paradigms: exact, approximate and heuristic. Exact algorithms are often based on solving an Integer Linear Program (ILP) using a combination of a Linear Programming (LP) relaxation and a branch-and-bound framework. Particularly for large problems, repeated solving of linear programs is computationally expensive and therefore prohibitive. Approximation algorithms address this concern, however, often at the expense of weak optimality guarantees. Moreover, approximation algorithms often involve manual construction for each problem. Seemingly easier to develop are heuristics which are generally computationally fast but guarantees are hardly provided. In addition, tuning of hyper-parameters for a particular problem instance may be required.

A fourth paradigm has been considered since the early 2000s and gained popularity again recently [93, 6, 85, 5, 27, 18]: learned algorithms. This fourth paradigm is based on the intuition that data governs the properties of the combinatorial algorithm. For instance, semantic image segmentation always deals with similarly sized problem structures or semantic patterns. It is therefore conceivable that learning to solve the problem on a given dataset uncovers strategies which are close to optimal but hard to find manually, since it is much more effective for a learning algorithm to sift through large amounts of sample problems. To achieve this, in a series of work, reinforcement learning techniques were developed [93, 6, 85, 5, 27, 18] and shown to perform well on a variety of combinatorial tasks from the traveling salesman problem and the knapsack formulation to maximum cut and minimum vertex cover.

While the aforementioned learning based techniques have been shown to perform extremely well on classical benchmarks, we are not aware of results for inference algorithms in CRFs for semantic segmentation. We hence wonder whether we can learn heuristics to address graphical model inference in semantic segmentation problems? To study this we develop a new framework for higher order CRF inference for the task of semantic segmentation using a Markov Decision Process (MDP). To solve the MDP, we assess two reinforcement learning algorithms: a Deep Q-Net (DQN) [58] and a deep net guided Monte Carlo Tree Search (MCTS) [82].

The proposed approach has two main advantages: (1) Unlike traditional approaches, it does not impose any constraints on the form of the CRF terms to facilitate effective inference. We demonstrate our claim by designing detection based higher order potentials that result in computationally intractable classical inference approaches. (2) Our method is more efficient than traditional approaches as inference complexity is linear in arbitrary potential orders while classical methods have exponential dependence on the largest clique size in general. This is due to the fact that semantic segmentation is reduced to sequentially inferring the labels.
of every variable based on a learned policy, without use of any iterative or search procedure.

We evaluate the proposed approach on two benchmarks: (1) the Pascal VOC semantic segmentation dataset [19], and (2) the MOTS multi-object tracking and segmentation dataset [86]. We demonstrate that our method outperforms traditional inference algorithms while being more efficient.

2. Related Work

We first review work on semantic segmentation before discussing learning of combinatorial optimizers.

Semantic Segmentation: In early 2000, classifiers were locally applied to images to generate segmentations [42] which resulted in a noisy output. To address this concern, as early as 2004, He et al. [33] applied Conditional Random Fields (CRFs) [43] and multi-layer perceptron features. For inference, Gibbs sampling was used, since MAP inference is NP-hard due to the combinatorial nature of the program. Progress in combinatorial optimization for flow-based problems in the 1990s and early 2000s [21, 23, 26, 9, 7, 8, 10, 40] showed that min-cut solvers can find the MAP solution of sub-modular energy functions of graphical models for binary segmentation. Approximation algorithms like swap-moves and α-expansion [10] were developed to extend applicability of min-cut solvers to more than two labels. Semantic segmentation was further popularized by combining random forests with CRFs [81]. Recently, the performance on standard semantic segmentation benchmarks like Pascal VOC 2012 [19] has been dramatically boosted by convolutional networks. Both deeper [48] and wider [61, 71, 92] network architectures have been proposed. Advances like spatial pyramid pooling [94] and atrous spatial pyramid pooling [13] emerged to remedy limited receptive fields. Other approaches jointly train deep nets with CRFs [16, 78, 28, 79, 52, 14, 96] to better capture the rich structure present in natural scenes.

CRF Inference: Algorithmically, to find the MAP configuration, LP relaxations have been extensively studied in the 2000s [74, 13, 41, 39, 22, 88, 34, 83, 35, 68, 89, 54, 53, 36, 75, 76, 77, 55, 56]. Also, CRF inference was studied as a differentiable module within a deep net [95, 51, 57, 24, 25]. However, both directions remain computationally demanding, particularly if high order potentials are involved. We therefore wonder whether recent progress in learning based combinatorial optimization yields effective algorithms for high order CRF inference in semantic segmentation.

Learning-based Combinatorial Optimization: Decades of research on combinatorial optimization, often also referred to as discrete optimization, uncovered a large amount of valuable exact, approximation and heuristic algorithms. Already in the early 2000s, but more prominently recently [93, 6, 85, 5, 27, 18], learning based algorithms have been suggested for combinatorial optimization. They are based on the intuition that instances of similar problems are often solved repeatedly. While humans have uncovered impressive heuristics, data driven techniques are likely to uncover even more compelling mechanisms. It is beyond the scope of this paper to review the vast literature on combinatorial optimization. Instead, we subsequently focus on learning based methods. Among the first is work by Boyan and Moore [6], discussing how to learn to predict the outcome of a local search algorithm in order to bias future search trajectories. Around the same time, reinforcement learning techniques were used to solve resource-constrained scheduling tasks [93]. Reinforcement learning is also the technique of choice for recent approaches addressing NP-hard tasks [5, 27, 18, 45] like the traveling salesman, knapsack, maximum cut, and minimum vertex cover problems. Similarly, promising results exist for structured predic-
At iteration $t$, $a_1 = (i_1^*, y_1^*)$ from the learned distribution $\pi(a_1 | s_1) \in \mathbb{R}^{N \times |\mathcal{L}|}$ at iteration $t = 1$.

Figure 2: Illustration of one iteration of reinforcement learning for the inference task. The policy network samples an action $a_1 = (i_1^*, y_1^*)$ from the learned distribution $\pi(a_1 | s_1) \in \mathbb{R}^{N \times |\mathcal{L}|}$ at iteration $t = 1$.

...learning based algorithms. In spirit, this mimics the design of heuristic rules. However, different from hand-crafting those rules, we use a learning based approach. To the best of our knowledge, this is the first work to successfully apply learning based combinatorial optimization to CRF inference for semantic segmentation. We therefore first provide an overview of the developed approach, outlined in Fig. 1.

Just like classical approaches, we also use local evidence and co-occurrence information, obtained from deep nets. This information is consequently used to form an energy function defined over a Conditional Random Field (CRF).

An example of a CRF with variables corresponding to super-pixels (circles), pairwise potentials (edges) and higher order potentials obtained from object detections (fully connected cliques) is illustrated in Fig. 1. However, different from classical methods, we find the minimizing configuration of the energy by repeatedly applying a learned policy network.

In every iteration, the policy network selects a random variable, i.e., the pixel and its label by computing a probability distribution over all currently unlabeled pixels and their labels.

Specifically, the pixel and label are determined by choosing the highest scoring entry in a matrix where the number of rows and columns correspond to the currently unlabeled pixels and the available labels respectively, as illustrated in Fig. 2.

3.2. Problem Formulation

Formally, given an image $x$, we are interested in predicting the semantic segmentation $y = (y_1, \ldots, y_N) \in \mathcal{Y}$. Hereby, $N$ denotes the total number of pixels or superpixels, and the semantic segmentation of a superpixel $i \in \{1, \ldots, N\}$ is referred to via $y_i \in \mathcal{L} = \{1, \ldots, |\mathcal{L}|\}$, which can be assigned one out of $|\mathcal{L}|$ possible discrete labels from the set of possible labels $\mathcal{L}$. The output space is denoted $\mathcal{Y} = \mathcal{L}^N$.

Classical techniques obtain local evidence $f_i(y_i)$ for every pixel or superpixel, and co-occurrence information in the form of pairwise potentials $f_{ij}(y_i, y_j)$ and higher order potentials $f_c(y_c)$. The latter assigns an energy to a clique $c \subseteq \{1, \ldots, N\}$ of variables $y_c = (y_i)_{i \in c}$. For readability, we drop the dependence of the energies $f_i, f_{ij}$ and $f_c$ on the image $x$ and the parameters of the employed deep nets. The goal of energy based semantic segmentation is to find the configuration $y^*$ which has the lowest energy $E(y)$, i.e.,

$$
y^* = \arg \min_{y \in \mathcal{Y}} E(y) \triangleq \sum_{i=1}^{N} f_i(y_i) + \sum_{(i,j) \in \mathcal{E}} f_{ij}(y_i, y_j) + \sum_{c \in \mathcal{C}} f_c(y_c).
$$

(1)

Hereby, the sets $\mathcal{E}$ and $\mathcal{C}$ subsume respectively the captured set of pairwise and higher order co-occurrence patterns. Details about the potentials are presented in Sec. 3.6.

Solving the combinatorial program given in Eq. (1), i.e., inferring the optimal configuration $y^*$ is generally NP-hard.
Different from existing methods, we develop a learning based combinatorial optimization heuristic for semantic segmentation with the intention to better capture the intricacies of energy minimization than can be done by hand-crafting rules. The developed heuristic sequentially labels one variable \(y_t, i \in \{1, \ldots, N\}\), at a time.

Formally, selection of one superpixel at a time can be formulated in a reinforcement learning context, as shown in Fig. 2. Specifically, an agent operates in \(t \in \{1, \ldots, N\}\) time-steps according to a policy \(\pi(a_t|s_t)\) which encodes a probability distribution over actions \(a_t \in \mathcal{A}_t\) given the current state \(s_t\). The current state subsumes in selection order the indices of all currently labeled variables \(I_t \subseteq \{1, \ldots, N\}\) as well as their labels \(y_{I_t} = (y_{i})_{i \in I_t}\), i.e., \(s_t \subseteq \{(I_t, y_{I_t}) : I_t \subseteq \{1, \ldots, N\}, y_{I_t} \in \mathcal{L}^{|I_t|}\}\). We start with \(s_1 = \emptyset\). The set of possible actions \(\mathcal{A}_t\) is the concatenation of the label spaces \(\mathcal{L}\) for all currently unlabeled pixels \(j \in \{1, \ldots, N\} \setminus I_t\), i.e., \(\mathcal{A}_t = \bigoplus_{(j \in \{1, \ldots, N\}) \setminus I_t} \mathcal{L}\). We emphasize the difference between the concatenation operator and the product operator used to obtain the semantic segmentation output space \(\mathcal{Y} = \mathcal{L}^N\), i.e., the proposed approach does not operate in the product space.

As mentioned before, the policy \(\pi(a_t|s_t)\) results in a probability distribution over actions \(a_t \in \mathcal{A}_t\) from which we greedily select the most probable action

\[ a_t^* = \arg \max_{a_t \in \mathcal{A}_t} \pi(a_t|s_t). \]

The most probable action \(a_t^*\) can be decomposed into the index for the selected variable, i.e., \(i_t^*\) and its state \(y_{i_t^*} \in \mathcal{L}\). We obtain the subsequent state \(s_{t+1}\) by combining the extracted variable index \(i_t^*\) and its labeling with the previous state \(s_t\). Specifically, we obtain \(s_{t+1} = s_t \oplus (i_t^*, y_{i_t^*})\) by slightly abusing the \(\oplus\)-operator to mean concatenation to a set and a list maintained within a state.

Formally, we summarize the developed reinforcement learning based semantic segmentation algorithm used for inferring a labeling \(\hat{y}\) in Alg. 1. In the following, we describe the policy function \(\pi_{\theta}(a_t|s_t)\), which we found to work well for semantic segmentation, and different variants to learn its parameters \(\theta\).

3.3. Policy Function

We model the policy function \(\pi_{\theta}(a_t|s_t)\) using a graph embedding network [17]. The input to the network is a weighted graph \(G(V, E, w)\), where nodes \(V = \{1, \ldots, N\}\) correspond to variables, i.e., in our case superpixels, \(E\) is a set of edges connecting neighboring superpixels, as illustrated in Fig. 1, and \(w : E \rightarrow \mathbb{R}^+\) is the edge weight function. The weights \(\{w(i, j)\}_{(i,j) \in E}\) on the edges between a given node \(i\) and its neighbors \(\{j : (i, j) \in E\}\) form a distribution, obtained by normalizing the dot product between the hypercolumns \([30]g_i\) and \(g_j\) via a softmax across neighbors. At every iteration, the state \(s_t\) is encoded in the graph \(G\) by tagging node \(i \in V\) with a scalar \(h_i = 1\) if the node is part of the already labeled set \(I_t\), i.e., if \(i \in I_t\) and 0 otherwise. Moreover, a one-hot encoding \(\hat{y}_i \in \{0, 1\}^{|\mathcal{L}|}\) encodes the selected label of nodes \(i \in I_t\). We set \(\hat{y}_i\) to equal the all zeros vector if node \(i\) has not been selected yet.

Every node \(i \in V\) is represented by a \(p\)-dimensional embedding, where \(p\) is a hyperparameter. The embedding is composed of \(\hat{y}_i, h_i\) as well as superpixel features \(h_i \in \mathbb{R}^p\) which encode appearance and bounding box characteristics that we discuss in detail in Sec. 4.

The output of the network is a \(|\mathcal{L}|\)-dimensional vector \(\pi_i\) for each node \(i \in V\), representing the scores of the \(|\mathcal{L}|\) different labels for variable \(i\).

The network iteratively generates a new representation \(\mu_i^{(k+1)}\) for every node \(i \in V\) by aggregating the current embeddings \(\mu_i^{(k)}\) according to the graph structure \(\mathcal{E}\) starting from \(\mu_i^{(0)} = 0, \forall i \in V\). After \(K\) steps, the embedding captures long range interactions between the graph features as well as the graph properties necessary to minimize the energy function \(E\). Formally, the update rule for node \(i\) is

\[
\mu_i^{(k+1)} = \text{Relu} \left( \theta_1^{(k)} h_i + \theta_2^{(k)} \hat{y}_i + \theta_3^{(k)} h_i + \theta_4^{(k)} \sum_{(i,j) \in E} w(i, j) \mu_j^{(k)} \right),
\]

where \(\theta_1^{(k)} \in \mathbb{R}^p, \theta_2^{(k)} \in \mathbb{R}^{p \times |\mathcal{L}|}, \theta_3^{(k)} \in \mathbb{R}^{p \times F}\) and \(\theta_4^{(k)} \in \mathbb{R}^{p \times p}\) are trainable parameters. After \(K\) steps, \(\pi_i\) for every unlabeled node \(i \in \{1, \ldots, N\} \setminus I_t\) is obtained via

\[
\pi_i = \theta_5 \mu_i^{(K)} \quad \forall i \in \{1, \ldots, N\} \setminus I_t,
\]

where \(\theta_5 \in \mathbb{R}^{S \times p}\) is another trainable model parameter. We illustrate the policy function \(\pi_{\theta}(a_t|s_t)\) and one iteration of inference in Fig. 2.

3.4. Reward Function:

To train the policy, ideally, the reward function \(r_t(s_t, a_t)\) is designed such that the cumulative reward coincides exactly with the objective function that we aim at maximizing, i.e., \(\sum_{t=1}^{T} r_t(s_t, a_t) = -E(\hat{y})\), where \(\hat{y}\) is extracted from \(s_{N+1}\). Hence, at step \(t\), we define the reward as the difference between the value of the negative new energy \(E_t\) and the negative energy from the previous step \(E_{t-1}\), i.e.,

\[
r_t(s_t, a_t) = E_{t-1}(y_{t-1}) - E_t(y_t), \quad E_0 = 0.
\]
tials depending on variables that are not labeled at time $t$ are not incorporated in the evaluation of $E_t(y_t)$. We also study a second scheme, where the reward is truncated to $+1$ or $-1$, i.e., $r_i(s_t, a_t) \in \{-1, 1\}$. For every selected node $i_t$, with label $y_{i_t}$, we compare the energy function $E_t(y_{i_t})$ with the one obtained when using all other labels $\hat{y}_t \in \mathcal{L} \setminus y_{i_t}$. If the chosen label $y_{i_t}$ results in the lowest energy, the obtained reward is $+1$, otherwise it is $-1$.

Note that the unary potentials result in a reward for every time step. Pairwise and higher order potentials result in a sparse reward as their value is only available once all the superpixels forming the pair or clique are labeled. We illustrate the energy and reward computation on a graph with three fully connected nodes in Tab. 1.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$i_t$</th>
<th>$E_t$</th>
<th>$r_t = -(E_t - E_{t-1})$</th>
<th>$r_t = \pm 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>$f_1(y_1)$</td>
<td>$-f_1(y_1)$</td>
<td>$-1+2\cdot \hat{1}(E_t(y_2) &lt; E_t(y_1) \vee y_1)$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>$f_1(y_1) + f_2(y_2) + f_{12}(y_1, y_2)$</td>
<td>$-f_2(y_2) - f_{12}(y_1, y_2)$</td>
<td>$-1+2\cdot \hat{1}(E_t(y_1, y_2) &lt; E_t(y_1, y_2) \vee y_2)$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>$f_1(y_1) + f_2(y_2) + f_{13}(y_1, y_3) + f_{1,2,3}(y_1, y_2, y_3)$</td>
<td>$-f_{13}(y_1, y_3) - f_{1,2,3}(y_1, y_2, y_3)$</td>
<td>$-1+2\cdot \hat{1}(E_t(y_1, y_2, y_3) &lt; E_t(y_1, y_2, y_3) \vee y_3)$</td>
</tr>
</tbody>
</table>

**Monte-Carlo Tree Search:** While DQN tries to learn a policy from looking at samples representing one action at a time, MCTS has the inherent ability to update its policy after looking multiple steps ahead via a tree search procedure. At training time, through extensive simulations, MCTS builds a statistics tree, reflecting an empirical distribution $\pi_{\text{MCTS}}(a_t|s_t)$. Specifically, for a given image, a node in the search tree corresponds to the state $s_t$ in our formulation and an edge corresponds to a possible action $a_t$. The root node is initialized to $s_1 = \emptyset$. The statistics stored at every node correspond to (1) $N(s_t)$: the number of times state $s_t$ has been reached, (2) $N(a_t|s_t)$: the number of times action $a_t$ was chosen in state $s_t$ in all previous simulations, as well as (3) $\hat{r}_t(s_t, a_t)$: the averaged reward across all simulations starting at $s_t$ and taking action $a_t$. The MCTS policy is defined as $\pi_{\text{MCTS}}(a_t|s_t) = \frac{N(a_t|s_t)}{N(s_t)}$. The simulations follow an exploration-exploitation procedure modeled by a variant of the Probabilistic Upper Confidence Bound (PUCB) [82]: $U(a_t, s_t) = \hat{r}_t(s_t, a_t) + \sqrt{\frac{1}{N(s_t)} \log N(s_t)}$. During exploration, we additionally encourage the same action sets $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ used for DQN. Also, similarly to DQN, the generated experiences $(s_t, \pi_{\text{MCTS}})$ are stored in a replay buffer. The policy network is then trained through a cross entropy loss

$$L(\theta) = -\sum_s \sum_a \pi_{\text{MCTS}}(a|s) \log \pi_\theta(a|s).$$

To obtain the empirically constructed distribution. Here, the second sum is over all valid actions from a state $s$ sampled from the replay buffer and $\pi_{\text{MCTS}}(a|s)$ is the corresponding empirically estimated distribution.

A more detailed description of the MCTS search process, including pseudo-code is available in Appendix B. At inference time, we use a low budget of simulations. Final actions are taken according to the constructed $\pi_{\text{MCTS}}(a_t|s_t)$.
The replay-memory for both DQN and MCTS is divided into two chunks. The first chunk corresponds to the unary potential, while the second chunk corresponds to the overall energy function. A node is assigned to the second chunk if its associated reward is higher than the one obtained from its unary labeling. This ensures positive rewards from all the potentials during training. Every chunk is further divided into $|L|$ categories corresponding to the $|L|$ classes of the selected node. This guarantees a balanced sampling of the label classes in every batch during training. Beyond DQN and MCTS, we experimented with policy gradients but could not get it to work as it is an on-policy algorithm. Reusing experiences for the structured replay buffer was crucial for success of the learning algorithm.

### 3.6. Energy Function

Finally we provide details on the energy function $E$ given in Eq. (1). The unary potentials $f_i(y_i) \in \mathbb{R}^{L}$ are obtained from a semantic segmentation deep net. The pairwise potential encodes smoothness and is computed as follows:

$$f_{i,j}(y_i, y_j) = \psi(y_i, y_j) \cdot \alpha_p \cdot |g_{i,j}| \cdot \beta_p,$$

where $\psi(y_i, y_j)$ is the label compatibility function describing the co-occurrence of two classes in adjacent locations and is given by the Potts model:

$$\psi(y_i, y_j) = \begin{cases} 1 & \text{if } y_i = y_j \\ 0 & \text{otherwise} \end{cases}.$$  

Moreover, $|g_{i,j}|$ is the above defined unnormalized weight $w(i, j)$ for the edge connecting the $i$th and $j$th nodes, i.e., superpixels. Intuitively, if the dot product between the hyperparameters $g_i$ and $g_j$ is smaller than a threshold $\beta_p$, and the two superpixels are labeled differently, a penalty of value $\alpha_p$ incurs.

While the pairwise term mitigates boundary errors, we address recognition errors with two detection-based higher order potentials [3]. For this purpose, we use the YoloV2 bounding box object detector [69] as it ensures a good trade-off between speed and accuracy. Every bounding box $b$ is presented by a tuple $(b_b, c_b, I_b)$, where $b_b$ is the class label of the detected object, $c_b$ is the confidence score of the detection and $I_b \subseteq \{1, \ldots, N\}$ is the set of superpixels that belong to the foreground detection obtained via Grab-Cut [72].

The first higher order potential (HOP1) encourages superpixels within a bounding box to take the bounding box label, while enabling recovery from false detections that do not agree with other energy types. For this purpose, we add an auxiliary variable $z_b$ for every bounding box $b$. We use $z_b = 1$, if the bounding box is inferred to be valid, otherwise $z_b = 0$. Formally,

$$f(y_{i,b}, z_b) = \begin{cases} w_b \cdot c_b \cdot \sum_{i \in I_b} 1_{y_i = l_i} & \text{if } z_b = 0 \\ w_b \cdot c_b \cdot \sum_{i \in I_b} 1_{y_i \neq l_i} & \text{if } z_b = 1 \end{cases}.$$  

This simplification enables solving the higher order potential using traditional techniques like mean field inference [3]. To show the merit of the RL framework, we introduce another higher order potential (HOP2) that can not be seamlessly reduced to a pairwise one:

$$f_i(y_{i,b}, z_b) = \begin{cases} w_b \cdot c_b \cdot 1_{y_i = l_i} & \text{if } z_b = 0 \\ w_b \cdot c_b \cdot 1_{y_i \neq l_i} & \text{if } z_b = 1 \end{cases}.$$  

This potential is evaluated for bounding boxes with special characteristics to encourage the superpixels in the bounding box to be of label $l$. Intuitively, if the number of superpixels $i \in I_b$ having label $l$ is less than a threshold $|l_b|$ \over{|I_b|}, a penalty $\lambda_b$ incurs. For Pascal VOC, we evaluate the potential on bounding boxes $b$ included in larger bounding boxes, as we noticed that the unaries frequently miss small objects overlapping with other larger objects in the image ($l = l_b$). For MOTS, we evaluate this potential on bounding boxes of type 'pedestrians' overlapping with bounding boxes of type 'bicycle.' As cyclists should not be labeled as pedestrians, we set $l_b$ to be the background class. Transforming this term into a pairwise one to enable using tradition inference techniques requires an inference number of auxiliary variables.

### Table 2: Performance results for the minimizing the energy function $E_t$ under reward scheme 1 ($R_t^2 = -(E_t - E_{t-1})$) and reward scheme 2 ($R_t^2 = \pm 1$).

<table>
<thead>
<tr>
<th>Method</th>
<th>Unary</th>
<th>Unary + Pairwise</th>
<th>Unary + Pairwise + HOP1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\psi$</td>
<td>$\alpha$</td>
<td>$\beta$</td>
</tr>
<tr>
<td></td>
<td>DQN MCTS</td>
<td>DQN MCTS</td>
<td>DQN MCTS</td>
</tr>
<tr>
<td></td>
<td>$\alpha_1$</td>
<td>$\alpha_2$</td>
<td>$\alpha_3$</td>
</tr>
<tr>
<td></td>
<td>$\beta_1$</td>
<td>$\beta_2$</td>
<td>$\beta_3$</td>
</tr>
<tr>
<td></td>
<td>$\gamma_1$</td>
<td>$\gamma_2$</td>
<td>$\gamma_3$</td>
</tr>
</tbody>
</table>

7594
In the following, we evaluate our learning-based inference algorithm on Pascal VOC [19] and MOTS [19] datasets. The original Pascal VOC dataset contains 1464 training and 1449 validation images. In addition to this data, we make use of the annotations provided by [29], resulting in a total of 10582 training instances. The number of classes is 21. MOTS is a multi-object tracking and segmentation dataset for cars and pedestrians (2 classes). It consists of 12 training sequences (5027 frames) and 9 validation ones (2981 frames). In this work, we perform semantic segmentation at the level of superpixels, generated using SLIC [84]. Every superpixel corresponds to a node \( i \) in the graph as illustrated in Fig. 1. The unary potentials at the pixel level are obtained from PSPNet [94] for Pascal VOC and TrackR-CNN [86] for MOTS. The superpixels’ unaries are the average of the unaries of all the pixels that belong to that superpixel. The higher order potential is based on the YoloV2 [69] bounding box detector. Additional training and implementation details are described in Appendix A.

**Evaluation Metrics:** As evaluation metrics, we use intersection over union (IoU) computed at the level of both superpixels (sp) and pixels (p). IoU (p) is obtained after mapping the superpixel level labels to the corresponding set of pixels.

**Baselines:** We compare our results to the segmentations obtained by five different solvers from three categories: (1) message passing algorithms, *i.e.*, Belief propagation (BP) [64] and Tree-reweighted Belief Propagation (TBP) [87], (2) a Lagrangian relaxation method, *i.e.*, Dual Decomposition Subgradient (DD) [37], and (3) move making algorithms, *i.e.*, Lazy Flipper (LFlip) [1] and \( \alpha \)-expansion as implemented in [20]. Note that these solvers cannot optimize our HOP2 potential. Besides, we train a supervised model that predicts the node label from the provided node features.

**Performance Evaluation:** We show the results of solving the program given in Eq. (1) in Tab. 2, for unary (Col. 5), unary plus pairwise (Col. 6), unary plus pairwise plus HOP1 (Col. 7) and unary plus pairwise plus HOP1 and HOP2 (Col. 8) potentials. For every potential type, we report results on graphs with superpixel numbers 50, 250 and 500 for Pascal VOC and 2000 for MOTS, obtained from DQN and MCTS, trained each with the two reward schemes discussed in Sec. 3.5. Since MOTS has small objects, we opt for a higher number of superpixels. It is remarkable to observe that DQN and MCTS are able to learn heuristics which outperform the baselines. Interestingly, the policy has learned to produce better semantic segmentations than the ones obtained via MRF energy minimization. Guided by a reward derived just from the energy function, the graph neural network (the policy) learns characteristic node embeddings for every semantic class by leveraging the hypercolumn and bounding box features as well as the neighborhood characteristics. The supervised baseline shows low performance, which proves the merit of the learned policies. Overall MCTS performance is comparable to the DQN one. This is mainly due to the learned policies being somewhat local and focusing on object boundaries, not necessitating a large multi-step look-ahead, as we will show in the following.

In Fig. 3, we report success cases of the RL algorithms. Smoothness modeled by our energy fixed the bottle segmentation in the first image. Furthermore, our model detects missing parts of the table in the second image in the first row and a car in the image in the second row, that were missed by the unaries. Also, we show that we fix a mis-labeling of a truck as a car in the image in the third row.

**Flexibility of Potentials:** In Fig. 4, we show examples of improved segmentations when using the pairwise, HOP1 and HOP2 potentials respectively. The motorcycle driver segmentation improved incrementally with every potential (first image) and the cyclist is not detected anymore as a pedestrian (second image).

**Generalization and scalability:** The graph embedding network enables training and testing on graphs with different number of nodes, since the same parameters are used. We investigate how models trained on graphs with few nodes perform on larger graphs. As shown in Tab. 3, compelling accuracy and IoU values for generalization to graphs with up to 500, 1000, 2000, and 10000 nodes are observed when using a policy trained on graphs of 250 nodes for Pascal VOC, and to graphs with up to 5000 and 10000 nodes when using...
a policy trained on 2000 nodes for MOTS. Here, we consider the energy consisting of the combined potentials (unary, pairwise, HOP1 and HOP2). Note that we outperform PSPNet at the pixel level for Pascal VOC.

**Runtime efficiency:** In Tab. 4, we show the inference runtime for respectively the baselines, DQN and MCTS. The runtime scales linearly with the number of nodes and does not even depend on the potential type/order in case of DQN, as inference is reduced to a forward pass of the policy network at every iteration (Fig. 2). DQN is faster than all the solvers apart from $\alpha$-exp. However, performance-wise, $\alpha$-expansion has worse results (Tab. 2). MCTS is slower as it performs multiple simulations per node and requires computation of the reward at every step.

**Learned Policies:** In Fig. 5, we show the probability map across consecutive time steps. The selected nodes are colored in white. The darker the superpixel, the smaller the probability of selecting it next. We found that the heuristic learns a notion of smoothness, choosing nodes that are in close proximity and of the same label as the selected ones. Also, the policy learns to start labeling the nodes with low unary distribution entropy first, then decides on the ones with higher entropy.

**Limitations:** Our method is based on super-pixels, hence datasets with small objects require a large number of nodes and a longer run-time (MOTS vs. Pascal VOC). Also, our method is sensitive to bounding box class errors, as illustrated in Fig. 6 (first example), and to the parameters calibration of the energy function, as shown in the second example of the same figure. We plan to address the latter concern in future work via end-to-end training. Furthermore, little is know about deep reinforcement learning convergence. Nevertheless, it has been successfully applied to solve combinatorial programs by leveraging the structure in the data. We show that in our case as well, it converges to reasonable policies.

**5. Conclusion**

We study how to solve higher order CRF inference for semantic segmentation with reinforcement learning. The approach is able to deal with potentials that are too expensive to optimize using conventional techniques and outperforms traditional approaches while being more efficient. Hence, the proposed approach offers more flexibility for energy functions while scaling linearly with the number of nodes and the potential order. To answer our question: can we learn heuristics for graphical model inference? We think we can but we also want to note that a lot of manual work is required to find suitable features and graph structures. For this reason we think more research is needed to truly automate learning of heuristics for graphical model inference. We hope the research community will join us in this quest.

**Acknowledgements:** This work is supported in part by NSF under Grant No. 1718221 and MRI #1725729, UIUC, Samsung, 3M, and Cisco Systems Inc. (Gift Award CG 1377144). We thank Cisco for access to the Arcetri cluster and Iou-Jen Liu for initial discussions.
References


