A Fast and Exact Energy Minimization Algorithm for Cycle MRFs

Huayan Wang HUAYANW@CS.STANFORD.EDU
Computer Science Department, Stanford University, Palo Alto, CA 94305 USA

Daphne Koller KOLLER@CS.STANFORD.EDU
Computer Science Department, Stanford University, Palo Alto, CA 94305 USA

Abstract

The presence of cycles gives rise to the difficulty in performing inference for MRFs. Handling cycles efficiently would greatly enhance our ability to tackle general MRFs. In particular, for dual decomposition of energy minimization (MAP inference), using cycle subproblems leads to a much tighter relaxation than using trees, but solving the cycle subproblems turns out to be the bottleneck.

In this paper, we present a fast and exact algorithm for energy minimization in cycle MRFs, which can be used as a subroutine in tackling general MRFs. Our method builds on junction-tree message passing, with a large portion of the message entries pruned for efficiency. The pruning conditions fully exploit the structure of a cycle. Experimental results show that our algorithm is more than an order of magnitude faster than other state-of-the-art fast inference methods, and it performs consistently well in several different real problems.

1. Introduction

We address the problem of energy minimization (MAP inference) in cycle MRFs. Specifically, consider a MRF over $\mathbf{X} = \{X_{1:N}\}$ with connectivity $X_1 - X_2 - \cdots - X_N - X_1$. It is parametrized by N pairwise potentials:

$$\boldsymbol{\Theta}(\mathbf{X}) = \sum_{i=1}^{N} \boldsymbol{\theta}_i \tag{1}$$

where $\boldsymbol{\theta}_i$ has the scope $\boldsymbol{\theta}_i(X_i, X_{i+1})$ (X_{N+1} is X_1). Our goal is to find assignments to $X_{1:N}$ that minimize the energy function (1).

Cycle MRFs could be useful as a modeling tool in its own right. For example, a cycle over landmarks could be used for top-down image segmentation (Heitz et al., 2009). More importantly, cycles have been recognized as a crucial component in tackling general MRFs (Werner, 2008; Sontag & Jaakkola, 2008; Komodakis & Paragios, 2008). In particular, a popular framework for solving the (NP-hard) MAP inference problem for general MRFs is to solve a LP relaxation of it by dual decomposition (decomposing it into a number of "easier" subproblems) (Komodakis et al., 2011; Sontag et al., 2011). The tightness of the underlying LP relaxation is determined by the choice of the subproblems. If all subproblems are tree-structured, the resulted LP relaxation is usually not tight for rich-structured MRFs. Researchers have been focusing on how to selectively add cycle subproblems in order to tighten the relaxation (Sontag et al., 2008; 2012; Batra et al., 2011). Solving a cycle MRF (usually a triplet cluster) is a crucial subroutine in both (1) evaluating the criterion for adding cycles/clusters and (2) computing subgradients/messages for optimizing the dual objective. However, it is usually the bottleneck procedure especially when the variable state spaces are large.

One way of solving the cycle MRF is by loopy max-product BP, which has shown to be exact (Weiss, 2000) for MRFs with a single cycle. Although each message passing step only takes $O(K^2)$ time (K is variable cardinality), the number of iterations needed for convergence is usually large, especially for small and "frustrated" cycles encountered in tightening dual decomposition. Another way of solving the cycle MRF (which we build on) is to pass messages in the clique tree (triangulated cycle), which takes $O(K^3)$ time if the messages are computed in the naive way. Faster

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

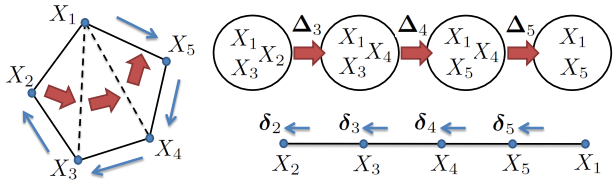

Figure 1. Messages passed in our fast cycle solver. **Wide red arrows:** messages in the clique tree (triangulated cycle). **Narrow blue arrows:** messages on cycle edges.

algorithms for computing the messages have been proposed recently (Felzenszwalb & McAuley, 2011; McAuley & Caetano, 2011); they still sequentially compute all full messages. However, we observe that only a tiny portion of the message entries need to be computed in order to find the MAP assignment—others can be pruned by conditions derived from the global structure of the cycle.

The pruning strategy in general has been used by existing work including search-based branch-and-bound methods (Marinescu & Dechter, 2006; 2007; Flerova et al., 2011; Sun et al., 2012), column generation (Belanger et al., 2012), and fast min-sum (Felzenszwalb & McAuley, 2011). We will elaborate on the relation between our algorithm and existing methods in Section 5. As our algorithm is specifically tailored to cycle MRFs, it is usually more than an order of magnitude faster than these general-purpose methods.

2. Fast Cycle Solver

To solve the cycle MRF we triangulate it in a special way (as in Fig. 1) such that X_1 is in the scope of all clique tree nodes. If we perform clique tree message passing from left to right, we would compute clique tree messages:

$$\boldsymbol{\Delta}^*_i(X_1, X_i) = \min_{X_{i-1}} [\boldsymbol{\Delta}^*_{i-1}(X_1, X_{i-1}) + \boldsymbol{\theta}_{i-1}(X_{i-1}, X_i)] \tag{2}$$

for $i = 3 : N$. (Note that $\boldsymbol{\Delta}^*_2 = \boldsymbol{\theta}_1$ in computing the first message.) They are shown by wide red arrows in Fig. 1.

Instead of computing the full messages $\boldsymbol{\Delta}^*_i$, we compute *incomplete* versions of them, denoted as $\boldsymbol{\Delta}_i$. To explain how these incomplete messages are computed we perceive each message $\boldsymbol{\Delta}_i$ as a *set* of all its entries:

$$\boldsymbol{\Delta}_i = \{\boldsymbol{\Delta}_i(x_1, x_i) \,|\, x_1 \in Val(X_1),\ x_i \in Val(X_i)\} \tag{3}$$

where $Val(X)$ is the domain of X. We use lowercase x to denote specific assignments to variable X.

Our algorithm will split the set into an *active* subset and an *inactive* one: $\boldsymbol{\Delta}_i = \boldsymbol{\Delta}^+_i \cup \boldsymbol{\Delta}^-_i$. Similarly we also split the original energy terms $\boldsymbol{\theta}_i$ into active and inactive subsets. Only the active parts participate in the *partial min-sum* operation (explained later). The notations are summarized in Table 1.

Table 1. Summary of notations in Section 2

$\boldsymbol{\theta}_i$	original energy terms (factors)
$\boldsymbol{\theta}^+_i$	active entries of $\boldsymbol{\theta}_i$
$\boldsymbol{\theta}^-_i$	inactive entries of $\boldsymbol{\theta}_i$
$\boldsymbol{\Delta}^*_i$	clique tree messages, from (2)
$\boldsymbol{\Delta}_i$	"incomplete" clique tree messages, from (4)
$\boldsymbol{\Delta}^+_i$	active entries of $\boldsymbol{\Delta}_i$
$\boldsymbol{\Delta}^-_i$	inactive entries of $\boldsymbol{\Delta}_i$
$\boldsymbol{\delta}_i$	messages on cycle edges

To define partial min-sum, we abbreviate (2) as $\boldsymbol{\Delta}^*_i = \text{minsum}(\boldsymbol{\Delta}^*_{i-1}, \boldsymbol{\theta}_{i-1})$, which defines the min-sum operation. Note that it can be performed (on two sets of entries) as follows:

1. For all $a := \boldsymbol{\Delta}^*_{i-1}(x_1, x_{i-1}) \in \boldsymbol{\Delta}^*_{i-1}$
2. For all $b := \boldsymbol{\theta}_{i-1}(x_{i-1}, x_i) \in \boldsymbol{\theta}_{i-1}$
3. $\boldsymbol{\Delta}^*_i(x_1, x_i) \leftarrow \min(\boldsymbol{\Delta}^*_i(x_1, x_i),\ a + b)$

Therefore it is straightforward to define partial min-sum:

$$\boldsymbol{\Delta}_i = \text{minsum}(\boldsymbol{\Delta}^+_{i-1}, \boldsymbol{\theta}^+_{i-1}) \tag{4}$$

by restricting the loops in the above algorithm to the active subsets.

At the end of the clique chain we compute

$$\widehat{\Theta} = \min(\boldsymbol{\Delta}_N + \boldsymbol{\theta}_N) \tag{5}$$

Note that $\boldsymbol{\Delta}_N$ and $\boldsymbol{\theta}_N$ have the same scope (X_N, X_1). Therefore we compute the sum and the min directly (without restricting to active subsets), and trace back to recover the corresponding assignment.

The general idea of our method is to repeatedly update all $\boldsymbol{\Delta}_i$ by gradually adding entries to each $\boldsymbol{\theta}^+_i$ and each $\boldsymbol{\Delta}^+_i$. This process stops when we are guaranteed (by the conditions given below) that we have obtained the MAP solution of the problem.

We will first describe the stopping conditions and then show the overall algorithm. To derive the stopping conditions we need to use another set of messages $\boldsymbol{\delta}_{2:N}$:

$$\boldsymbol{\delta}_i(X_i) = \min_{X_{i+1}} [\boldsymbol{\delta}_{i+1}(X_{i+1}) + \boldsymbol{\theta}_i(X_i, X_{i+1})] \tag{6}$$

where $\boldsymbol{\delta}_{N+1} = \mathbf{0}$ for $i = N$. These messages are passed along the cycle edges (shown by the narrow blue arrows in Fig. 1). Note that unlike in loopy BP, $\boldsymbol{\delta}_{2:N}$ do *not* form a loop, so they are computed once and for all. Computing them only has the time complexity of $O(K^2)$.

Suppose that we are in the process of iteratively updating the incomplete messages. Given current $\boldsymbol{\Delta}_{3:N}$ and the active/inactive split of $\boldsymbol{\Delta}_{3:N-1}$ and $\boldsymbol{\theta}_{1:N-1}$, we assume that each incomplete message has been computed according to (4), and we have the current best solution with energy $\widehat{\Theta}$ computed by (5). The optimality of $\widehat{\Theta}$ can be established by the following conditions.

Proposition 1. *$\widehat{\Theta}$ is the global minimum of the cycle MRF energy if all entries $\boldsymbol{\theta}_i(x_i, x_{i+1}) \in \boldsymbol{\theta}_i^-$, for $i = 1 : N-1$, satisfy:*

$$\boldsymbol{\theta}_i(x_i, x_{i+1}) > \widehat{\Theta} - \min_{X_1} \boldsymbol{\Delta}_i(X_1, x_i) - \boldsymbol{\delta}_{i+1}(x_{i+1}) \quad (7)$$

and all entries $\boldsymbol{\Delta}_i(x_1, x_i) \in \boldsymbol{\Delta}_i^-$, for $i = 3 : N-1$, satisfy:

$$\boldsymbol{\Delta}_i(x_1, x_i) > \widehat{\Theta} - \boldsymbol{\delta}_i(x_i) \quad (8)$$

Proof is given in supplement material. Here we give some intuitive interpretations.

Given an entry in $\boldsymbol{\theta}$ or $\boldsymbol{\Delta}$ we want to judge whether or not it could possibly lead to a lower energy than $\widehat{\Theta}$. If not, we can safely leave it in the inactive set. Ideally, we would like a tight bound, i.e. all entries except for the ones corresponding to the MAP assignment satisfy (7) and (8). Now the question is, how far are we from that ideal bound? It turns out that we are only off by the uncertainty on one variable X_1. Specifically, for (7), note that the term $\min_{X_1} \boldsymbol{\Delta}_i(X_1, x_i)$ involves a choice in X_1, and the term $\boldsymbol{\delta}_{i+1}(x_{i+1})$ also (implicitly) involves a choices in X_1, because it has minimized over X_1 in the beginning the $\boldsymbol{\delta}$ messages. If we somehow know that these two terms agree on their choices of X_1, it can be shown that we have the ideal bound, i.e. all entries except for the ones corresponding to the MAP assignment satisfy (7). We will make this precise by a corollary in supplement material.

An analogous argument can be made for the two terms $\boldsymbol{\Delta}_i(x_1, x_i)$ and $\boldsymbol{\delta}_i(x_i)$ in (8).

Indeed, the connectivity on one variable is how the cycle differs from a chain. In this regard, the conditions (7) and (8) fully exploit the structure of a cycle.

Now we come to the overall structure of the algorithm. The general idea is to gradually put small batches of entries into the active sets (using gradually increasing thresholds), and performing the partial min-sums iteratively. Each pass through the clique chain will update $\widehat{\Theta}$ and therefore the RHS of the conditions (7) and (8) will become smaller. Once these decreasing bounds meet the increasing thresholds, we are guaranteed that the current solution (with energy $\widehat{\Theta}$) is optimal. Our algorithm is sketched in Alg. 1. The strategy of using an increasing threshold to update messages has also been used in (Felzenszwalb & McAuley, 2011). In practice we offset all potentials to be non-negative, so we can start with a small positive number as threshold and increase it by multiplying with some constant.

Algorithm 1 Algorithm sketch of our fast cycle solver

```
T_{3:N} ← ε
compute δ_{2:N}
while conditions (7,8) are not met do
  for i = 3 : N do
    while no updates is made to Δ_i do
      if T_i < Θ̂ − min δ_i then
        T_i ← min{1.5 × T_i, Θ̂ − min δ_i}
      Add entries ≤ min(T_i, RHS(7)) to θ^+_{i−1}.
      Add entries ≤ min(T_i, RHS(8)) to Δ^+_{i−1}.
      Perform partial min-sum (4). Update Δ_i (and
      Θ̂ if i = N)
```

3. Solving Cycles in Dual Decomposition

Consider energy minimization for general MRFs using dual decomposition. For brevity we only give a minimum introduction to the background, the general framework and implications have been thoroughly discussed in (Komodakis et al., 2011; Sontag et al., 2011).

The general idea is to decompose the MRF into subproblems $\{\boldsymbol{\Theta}^c\}$ (e.g., trees, cycles), where c indexes subproblems. Summing over $\{\boldsymbol{\Theta}^c\}$ gives a reparametrization of the original MRF energy. Presumably the subproblems are easy to solve, and each subproblem chooses its own assignment to the variables. We aim at maximizing the dual objective:

$$\mathcal{D}(\boldsymbol{\Theta}^c(\boldsymbol{X})) = \sum_c \min_{\boldsymbol{X}} (\boldsymbol{\Theta}^c(\boldsymbol{X})) \quad (9)$$

which lower bounds the minimum energy of the original MRF. Optimizing the dual objective can be viewed as exchanging messages for mutual agreements among subproblems (Wang & Koller, 2013). Maximizing (9) turns out to be the dual of some underlying LP relaxation of the original energy

minimization problem, whose tightness depends on the choice of subproblems in the decomposition.

For rich structured models we usually have to add cycle subproblems to the decomposition to tighten the underlying LP relaxation. In such a scenario we need a cycle solver in three situations:

1. To evaluate the criterion in selecting cycles to tighten the relaxation, such as the criterion in (Sontag et al., 2008).

2. To compute subgradients for optimizing the dual, such as in (Komodakis et al., 2011).

3. To compute messages for optimizing the dual (Wang & Koller, 2013), such as in MSD (Werner, 2007), MPLP (Globerson & Jaakkola, 2007) and TRW-S (Kolmogorov, 2006).

In the first two cases our algorithm can be directly plugged in. The third case has more subtlety, which we will focus on in this section.

To compute the messages in dual algorithms listed in the third case above, we usually need *min-marginals* of the cycle subproblems. The min-marginal over one variable X_i is defined as:

$$\mathbf{M}^c_{X_i}(x_i) = \min_{X_i = x_i} \boldsymbol{\Theta}^c(\mathbf{X}) \tag{10}$$

And we can define $\mathbf{M}^c_{(X_i,X_j)}$ for an edge analogously.

However, our algorithm does not compute the min-marginals. In the following we show a heuristic method to address this issue, which works well in practice as we demonstrate in experiments. However, it remains an interesting open problem to design more principled and efficient dual methods building on the fast cycle solver.

3.1. Sparse updates in dual methods

We consider the dual method of (Kolmogorov, 2006)[1], which iterates over nodes/edges of the MRF and, for each node/edge, average the min-marginals of all subproblems sharing that node/edge. Specifically, if X is selected as the node to be updated, the dual updates are:

$$\boldsymbol{\theta}^c_X(x) \leftarrow \boldsymbol{\theta}^c_X(x) + \overline{\mathbf{M}}_X(x) - \mathbf{M}^c_X(x), \tag{11}$$

for $\forall c \in \mathcal{J}(X)$, $\forall x \in Val(X)$, where $\mathcal{J}(X)$ denotes all subproblems sharing X. The updates are applied to $\boldsymbol{\theta}^c_X$, the potential of subproblem c on X. And

[1]The sparse dual updates introduced here can be applied to other dual methods similarly.

Table 2. Summary of notations in Section 3

$\mathcal{J}(X)$	set of subproblems containing variable X
$\boldsymbol{\Theta}^c$	energy function of subproblem c
$\boldsymbol{\theta}^c_X$	potential of subproblem c on variable X
$\mathbf{M}^c_X$	min-marginal of subproblem c on variable X
$\overline{\mathbf{M}}_X$	average of $\mathbf{M}^c_X$ over $c \in \mathcal{J}(X)$
$\widetilde{\mathbf{M}}_X$	average of "normalized" $\mathbf{M}^c_X$ over $c \in \mathcal{J}(X)$
$\widetilde{\mathbf{M}}^c_X$	upper bound approximation of $\mathbf{M}^c_X$
x^*_c	minimizer of $\mathbf{M}^c_X$ (and $\widetilde{\mathbf{M}}^c_X$)
$\mathcal{L}$	subset of $Val(X)$ with high priority (13)

$\overline{\mathbf{M}}_X = \frac{\sum_{c\in\mathcal{J}(X)} \mathbf{M}^c_X}{|\mathcal{J}(X)|}$ is the average min-marginal. The updates monotonically increase the dual objective (9). To compute the dual updates we need min-marginals from each subproblem. Table 2 summarizes notations used in this section.

In our cycle solver we have computed $\boldsymbol{\Delta}_N + \boldsymbol{\theta}_N$ in (5), which would give us the min-marginal $\mathbf{M}_{(X_N,X_1)}$ if we were using the complete message $\boldsymbol{\Delta}^*_N$. With the incomplete message we get an element-wise upper bound of the min-marginal, denoted by:

$$\widetilde{\mathbf{M}}_{(X_N,X_1)} = \boldsymbol{\Delta}_N + \boldsymbol{\theta}_N \tag{12}$$

And we can easily compute $\widetilde{\mathbf{M}}_{X_N}$ and $\widetilde{\mathbf{M}}_{X_1}$ from it. As the cycle is symmetric, this can be computed for any edge and node.

Directly using these upper-bound approximations in dual methods does not work well, because the resulted dual updates (messages) tend to overshoot, causing large oscillation in the dual objective. However, these approximate min-marginals bear important information that we can exploit.

Specifically, let us denote $x^* = \arg\min \widetilde{\mathbf{M}}_X(x)$. For any other assignment x, the difference $\widetilde{\mathbf{M}}_X(x) - \widetilde{\mathbf{M}}_X(x^*)$ is an upper bound approximation of the actual difference $\mathbf{M}_X(x) - \mathbf{M}_X(x^*)$ (because we know that $\widetilde{\mathbf{M}}_X(x^*) = \mathbf{M}_X(x^*) = \widehat{\Theta}$ from Proposition 1). This difference is an indication of the "relevance" of label x. The smaller the difference, the more likely that x will become optimal (for that cycle subproblem) and therefore affect the dual objective in subsequent dual updates.

To make use of this information, given X as the next node to be updated, we select an *active label set* $\mathcal{L} \subset Val(X)$ as the assignments with highest priority defined as:

$$\text{priority}(x) = -\min_{c\in\mathcal{J}(X)} (\widetilde{\mathbf{M}}^c_X(x) - \widetilde{\mathbf{M}}^c_X(x^*_c)), \tag{13}$$

where x^*_c is the optimal label for subproblem c.

Given the selected label set $\mathcal{L}$, we perform the dual updates (11) only for $x \in \mathcal{L}$. So we only need min-marginals $\mathbf{M}_X^c(x)$ for a sparse subset of labels. Each $\mathbf{M}_X^c(x)$ can be computed exactly by conditioning on $X = x$ and performing inference on the resulted chain MRF. In practice we choose $|\mathcal{L}|$ to be a small number to balance the effectiveness of dual updates and the extra cost of computing $\mathbf{M}_X^c(x)$.

When the dual updates (11) are applied to a subset $\mathcal{L}$ instead of $Val(X)$, monotonicity in the dual is no longer guaranteed. This can be fixed by "normalizing" the dual updates as follows:

Proposition 2. *Given X, if we perform sparse dual updates for all $c \in \mathcal{J}(X)$ and $x \in \mathcal{L} \subset Val(X)$, the dual objective (9) increases monotonically under* ***normalized*** *updates (14), but not under (11).*

$$\boldsymbol{\theta}_X^c(x) \leftarrow \boldsymbol{\theta}_X^c(x) + \widehat{\mathbf{M}}_X(x) + \mathbf{M}_X^c(x_c^*) - \mathbf{M}_X^c(x), \quad (14)$$

where $\widehat{\mathbf{M}}_X(x) = \frac{\sum_{c \in \mathcal{J}(X)} (\mathbf{M}_X^c(x) - \mathbf{M}_X^c(x_c^*))}{|\mathcal{J}(X)|}$.

Proof is given in supplement material. Intuitively we just replaced each min-marginal with its "normalized" version (subtracted the minimum energy of that subproblem). This maintains the monotonicity of the dual updates when performed on a subset of labels.

It worth noting that monotonicity is not sufficient for converging to some desired dual state (such as weak tree agreement (Kolmogorov, 2006)). Specifically, we observe that if we only include currently optimal labels in $\mathcal{L}$, the algorithm would easily get stuck. However, our selection criterion (13) also includes many non-optimal labels into $\mathcal{L}$. In our experiments the sparse update (14) always converges to dual optimal whenever the full TRW-S update (11) is able to converge.

All the above discussion was for updating node X. It can be easily generalized to updating edges as well, in which case $\mathcal{L}$ would be a subset of the joint assignments $Val(X_i) \times Val(X_j)$.

4. Experimental Results

All experiments are conducted using a single thread on a 3.3GHz CPU and 16 Gigabytes of memory.

4.1. Synthetic cycle MRFs

First we evaluate our cycle solver on synthetic cycle MRFs to compare it with various fast inference algorithms. In Fig. 2, we show how running time scales with the state space size K and number of variables N, respectively. For each setting, running time is averaged over 20 problem instances generated by sampling all edge potentials from $\mathcal{N}(0, 1)$. All methods are plotted with error bars indicating standard deviation across problem instances.

We can see that adapting to the problem structure improves performance. *AOBB* (Marinescu & Dechter, 2006) is a completely general-purpose algorithm[2], thus it is not surprising that it does not beat specialized methods. (Indeed, it is in some sense unfair to include it in the comparison.) Naive and fast min-sums (Felzenszwalb & McAuley, 2011) are tailored for tree-width-two graphs[3]. *Branch-and-Bound* (Sun et al., 2012) is designed for large state spaces. Our method fully exploits the structure of a cycle. It outperforms all other methods by more than an order of magnitude. (Note that the time axis is in log scale.)

We also observe (from Fig. 2 right) that the advantage of our cycle solver is especially prominent in solving triangles ($N = 3$). Note that in dual decomposition, we could triangulate any "untight" cycle and only handle triangle subproblems. The trade off between having fewer subproblems (large cycles) and having smaller subproblems (triangles) deserves further investigation.

4.2. Cycle subproblems in dual decomposition

To evaluate our cycle solver in dual decomposition, we used four MAP inference tasks as described below.

Synthetic Grid. We generated a five-by-five grid MRF (four-neighbors connectivity) with 25 variables and 1,000 states for each variables. All potentials are drawn from $\mathcal{N}(0, 1)$.

Pose Estimation. The task is to estimate the image positions of six upper-body parts of a human depicted in a single image. Each part is a variable, whose state space consists of discrete image positions pre-selected by some detector. The MRF is fully connected (with 15 edges), and the edge potentials capture spatial consistency of the parts. We used the problem instances provided in the software package of (Sun et al., 2012). The variable cardinality varies from 269 to 750.

SIFT Matching. We formulate the problem of matching two sets of image feature as MAP inference. Each feature point in the source image is a variable, its

[2] We also tried to run AND/OR graph best-first search (Marinescu & Dechter, 2007). It performs no better than *AOBB* in this task.

[3] It has been shown that (Felzenszwalb & McAuley, 2011) outperforms (McAuley & Caetano, 2011) in this task, so we did not include the latter into comparison.

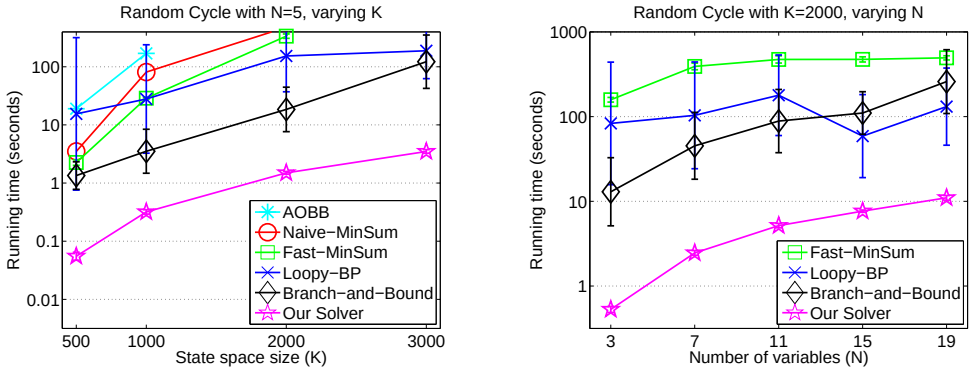

Figure 2. Comparing Cycle Solvers. **Note that time axis is in log scale.** *AOBB* is AND/OR graph branch-and-bound (Marinescu & Dechter, 2006). *Naive-MinSum* is the naive way of passing clique tree messages. *Fast-MinSum* is algorithm 2 in (Felzenszwalb & McAuley, 2011). *Loopy-BP* is loopy min-sum message passing. Sometimes it takes extremely long to converge, especially for short cycles. We always cut it off at 500 seconds. *Branch-and-Bound* is the method of (Sun et al., 2012).

state space is all feature points in the target image[4]. The problem instance we constructed has 68 variables, each has 1,021 states. The unary potentials measure similarity between the SIFT feature descriptors (Lowe, 2004). The MRF is constructed by connecting each source image feature point to its 3-nearest-neighbors (according to Euclidean distance, in source image). The resulted graph has many cycles. Pairwise potentials measure spatial consistency of two assignments[5].

Protein Design. The protein design benchmark (Yanover et al., 2006) features very densely connected MAP-MRF instances. Each MRF typically has hundreds of variables, thousands of edges, tens of thousands of triplet cycles. The cardinality of variables range from 2 to 180.

Experiments on these MRF inference problems focus on evaluating our method in two aspects:

1. Does the sparse dual update strategy (Section 3.1) optimize the dual effectively? Does the speed advantage of our fast cycle solver overcome the extra cost of filling-in the sparse min-marginals?

2. How does our cycle solver compare to other methods in solving the cycle subproblems encountered in dual decomposition? They may have different properties from the standalone synthetic cycles.

We address each of them below.

Comparing Dual Objectives. Among all cycle solvers we compared in Section 4.1, Fast-MinSum (Felzenszwalb & McAuley, 2011) is the fastest one that provides exact min-marginals, for which there is no need to use sparse dual updates as in Section 3.1). Therefore we focus on comparing Fast-MinSum and our cycle solver.

In Fig. 3, we show primal-dual plots for each of the four tasks. The three methods been compared are:

1. **Tree Decomposition**: Decompose the MRF into all its constituent edges. It has the same duality gap as decomposing into larger trees. The latter has no speed advantage unless the MRF structure allows us to construct large " monotonic chains" (Kolmogorov, 2006). This usually has a large duality gap for rich-structured models[6].

2. **Cycle Decomposition Fast-MinSum**: Use a fixed number of cycle subproblems, and fill-in uncovered edges (if there is any) using edge subproblems[7]. For Synthetic Grid we use all 16 quads. For Pose Estimation and SIFT Matching we use all triangles. For Protein Design we use 2,000 cycles selected using the criterion of (Sontag et al., 2008). Fast-MinSum is used to solve cycles.

[4]In practice one might want to disallow many-to-one mapping, or allow outliers in source image (that map to nothing). We ignore these issues here for simplicity.

[5]We measure spatial consistency by $\exp(-(d_0 - d_1)^2)$, where d_0 is the distance between the two source image points; d_1 is the distance between the two target image points.

[6]However, tree decomposition turns out to be tight for all Pose Estimation instances from (Sun et al., 2012). This is because the unary potentials are overwhelmingly strong comparing to the pairwise potentials. This is not the case for most fully connected MRFs.

[7]We first run tree-decomposition to convergence and switch to the cycle mode, except for the Pose Estimation case, where we directly start with the cycle mode.

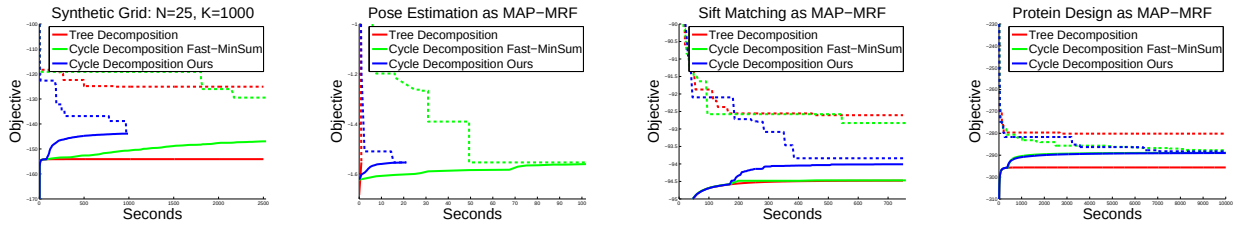

Figure 3. Primal-dual objectives in four MRF energy minimization tasks. See text for explanation.

Table 3. Average time (seconds) for solving one cycle subproblem in dual decomposition. Loopy-BP is cut off at 200 seconds if not converging.

	K	Fast-MinSum	Loopy-BP	Branch-and-Bound	Ours
Synthetic Grid	1000	16.4 (55 ×)	200 (667 ×)	33.2 (111 ×)	**0.30**
Pose Estimation	269∼750	0.54 (18 ×)	66.7 (2,223 ×)	0.18 (6 ×)	**0.03**
Sift Matching	1021	14.0 (93 ×)	87.8 (585 ×)	0.88 (5.9 ×)	**0.15**
Protein Design	2∼180	0.0023 (3.8 ×)	88.9 (14,574 ×)	0.28 (459 ×)	**0.00061**

3. **Cycle Decomposition Ours**: Same as the previous one except that our cycle solver and sparse dual updates[8] are used. We observe that this converges much faster than using Fast-MinSum in the first three cases. In the fourth case (Protein Design) the performances are comparable. This is because the variable cardinality in this case is much smaller, and the speed advantage of our cycle solver is not as large as in the other cases.

For the tasks with multiple problem instances (Protein Design and Pose Estimation), we only show the primal-dual plot of one instance. Different problem instances are qualitatively very similar.

Comparing Cycle Solver Time. We compare our cycle solver with three methods (that appeared more competitive in Section 4.1) on cycle subproblems encountered in dual decomposition. For each of the four tasks we sample about a hundred cycle instances, which span all stages of optimizing the dual. In Table 3 we show the average running time for solving a cycle, as well as its ratio with our cycle solver. We observe that Loopy-BP performs very badly in these usually "frustrated" cycles encountered in dual decomposition. Fast-MinSum and Branch-and-Bound may work well in one case but not so good in another. This could be because they are sensitive to different properties of the potentials in different problems. Our cycle solver works consistently well in all cases.

[8] We set $|\mathcal{L}|$ to be $\frac{1}{10}$ of variable cardinality in all cases.

4.3. Effect of cycle reparametrization

In this part we further analyze the behavior of our cycle solver. Notably it does not have a preferable worst-case guarantee (just like most branch-and-bound or pruning based methods). So one might ask: in what situations does our method work unsatisfactorily, and how to avoid those situations?

We do not have a conclusive theoretical characterization at this point. But we will show some empirical results to provide some insights on this issue.

Intuitively, our algorithm works better if the stopping criteria in Proposition 1 are tighter, which requires different parts of the cycle having more "mutual agreement". So we consider a local reparametrization to increase/decrease mutual agreement among edge potentials. Specifically, consider the two edge potentials $\boldsymbol{\theta}_1(X_1, X_2)$ and $\boldsymbol{\theta}_N(X_N, X_1)$. Let $\boldsymbol{\eta}_1 = \min_{X_2} \boldsymbol{\theta}_1$ and $\boldsymbol{\eta}_N = \min_{X_N} \boldsymbol{\theta}_N$. Note that both $\boldsymbol{\eta}$ are unary tables of X_1. Consider the following updates (denoted as $\Omega\pm$):

$$\boldsymbol{\theta}_1 \leftarrow \boldsymbol{\theta}_1 \pm \frac{(\boldsymbol{\eta}_N - \boldsymbol{\eta}_1)}{2}, \quad \boldsymbol{\theta}_N \leftarrow \boldsymbol{\theta}_N \pm \frac{(\boldsymbol{\eta}_1 - \boldsymbol{\eta}_N)}{2} \quad (15)$$

Note that $\Omega+$ will increase mutual agreement, whereas $\Omega-$ will decrease mutual agreement (between $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_N$).

To see the effect of these reparametrizations, we repeatedly apply $\Omega+$ (or $\Omega-$) to randomly selected nodes in the cycle, and measure the running time of different cycle solvers. Fig. 4 shows the running time (and its standard deviations) of three methods under repeated applications of $\Omega+$ (or $\Omega-$) to the cycle (-20 means applying $\Omega-$ to randomly selected nodes 20

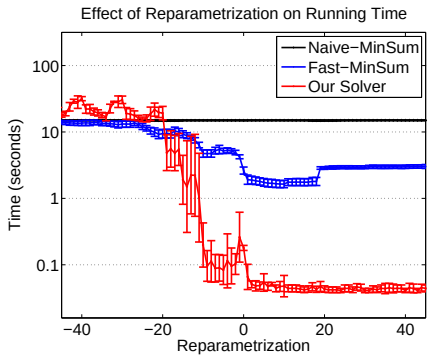

Figure 4. Effect of reparametrization. See text for explanation

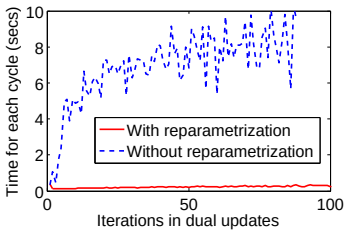

Figure 5. In the Synthetic Grid experiment, applying $\Omega+$ has a great impact on the cycle solver running time when the cycle potentials are iteratively updated in dual decomposition. See text for more explanation.

times). The problem instances are randomly generated (as in Section 4.1) with $N = 10$, $K = 500$. We observe that the negative reparametrizations have no influence on Naive-MinSum, small influence on Fast-MinSum, and very large influence on our method. This experiment characterizes the situation that our method needs to avoid, and also suggests a simple heuristic to get out of it: using $\Omega+$.

In dual decomposition, when solving cycle subproblems under repeated dual updates. The updates tend to "pull" different edges of the cycle away from each other (a similar effect as $\Omega-$). To tackle this, we apply $\Omega+$ once for each node of the cycle before running our cycle solver. This is very effective as shown in Fig. 5. Note that the extra cost of applying $\Omega+$ has been included. This heuristic was used in all experiments of Section 4.2.

5. Related Work

Our method is closely related to the fast min-sum (Felzenszwalb & McAuley, 2011). The key difference is that, fast min-sum computes each min-sum message in isolation, and prunes out entries that are *not necessary in computing that (full) message*; our method updates all min-sum messages iteratively, and prunes out entries that are *not necessary in computing the MAP assignment*. Our strategy exploits the global structure of the cycle and prunes out much more entries.

Comparing to traditional branch-and-bound methods (e.g., (Sun et al., 2012)), our algorithm is different in two aspects: (1) We do not explicitly maintain branches. Indeed, the message entries implicitly define a large number of overlapping "branches". (2) The "bound" in our method (Proposition 1) fully exploits the structure of a cycle, thus is much tighter than that derived from general principles.

Pruning in message passing has also been implemented using column generation (Belanger et al., 2012), which is very different from our approach. First of all, (Belanger et al., 2012) handles chains instead of cycles. Even if we generalize it by applying the same principles to the "clique chain" (triangulated cycle), there are still two key differences. (1) Since column generation is equivalent to cutting plane in the dual, the messages (dual variables) upon convergence need to be dual feasible. However, when our stopping criteria are met, the (incomplete) messages are generally *not* dual feasible, i.e. our stopping criterion is tighter than that derived from the general principles of column generation. (2) Moreover, the efficiency of column generation oracle relies on data-independent terms (precomputed bounds on the transitions), whereas our approach does not have such constraints.

The standard LP relaxation can also be tightened without explicitly solving cycle MRFs. For example (Komodakis & Paragios, 2008) proposed to *repair cycles*, which resembles subgradient updates (Komodakis et al., 2011) in dual decomposition—they both focus on updating dual variables associated with currently optimal labels. Note that it does not circumvent the intrinsic cubic complexity of cycle MRFs, because each cycle may need to be repaired with respect to all labels (anchor nodes) for one variable, and each repairing operation is quadratic in the state space size. Sontag et al. (2009) proposed to partition the state spaces and only enforce consistency at the coarse level. Yarkony et al. (2011) proposed to use binary cycle subproblems constructed by partitioning the state spaces. These binary cycles enforce fewer constraints, so we may need a large number of them for one cycle in the graph.

Acknowledgement

This work has been supported by the Max Planck Center for Visual Computing and Communication.

References

Batra, Dhruv, Nowozin, Sebastian, and Kohli, Pushmeet. Tighter relaxations for MAP-MRF inference: A local primal-dual gap based separation algorithm. *Journal of Machine Learning Research - Proceedings Track*, 15:146–154, 2011.

Belanger, David, Passos, Alexandre, Riedel, Sebastian, and McCallum, Andrew. Map inference in chains using column generation. In Bartlett, P., Pereira, F.C.N., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1853–1861. 2012.

Felzenszwalb, P. F. and McAuley, J. J. Fast inference with min-sum matrix product. *IEEE Trans. Pattern Anal. Mach. Intell*, 33(12):2549–2554, 2011.

Flerova, Natalia, Ihler, Alexander, Dechter, Rina, and Otten, Lars. Mini-bucket elimination with moment matching. In *NIPS Workshop DISCML*, 2011.

Globerson, Amir and Jaakkola, Tommi. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *NIPS*, 2007.

Heitz, G., Elidan, G., Packer, B., and Koller, D. Shape-based object localization for descriptive classification. *International Journal of Computer Vision*, 84(1):40–62, 2009.

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

Komodakis, N. and Paragios, N. Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles. In *ECCV*, pp. III: 806–820, 2008.

Komodakis, Nikos, Paragios, Nikos, and Tziritas, Georgios. MRF energy minimization and beyond via dual decomposition. *IEEE Trans. Pattern Anal. Mach. Intell*, 33(3):531–552, 2011.

Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60 (2):91–110, 2004.

Marinescu, Radu and Dechter, Rina. Memory intensive branch-and-bound search for graphical models. In *AAAI*, pp. 1200–1205. AAAI Press, 2006.

Marinescu, Radu and Dechter, Rina. Best-first AND/OR search for graphical models. In *AAAI*, pp. 1171–1176. AAAI Press, 2007.

McAuley, J. J. and Caetano, T. S. Faster algorithms for max-product message-passing. *Journal of Machine Learning Research*, 12(4):1349–1388, 2011.

Sontag, David and Jaakkola, Tommi. New outer bounds on the marginal polytope. In *NIPS*, pp. 1393–1400, 2008.

Sontag, David, Meltzer, Talya, Globerson, Amir, Jaakkola, Tommi, and Weiss, Yair. Tightening lp relaxations for map using message passing. In *UAI*, pp. 503–510, 2008.

Sontag, David, Globerson, Amir, and Jaakkola, Tommi. Clusters and coarse partitions in LP relaxations. In *NIPS*, pp. 1537–1544, 2009.

Sontag, David, Globerson, Amir, and Jaakkola, Tommi. Introduction to dual decomposition for inference. In Sra, Suvrit, Nowozin, Sebastian, and Wright, Stephen J. (eds.), *Optimization for Machine Learning*. MIT Press, 2011.

Sontag, David, Choe, Do Kook, and Li, Yitao. Efficiently searching for frustrated cycles in map inference. In *UAI*, 2012.

Sun, Min, Telaprolu, Murali, Lee, Honglak, and Savarese, Silvio. Efficient and exact MAP-MRF inference using branch and bound. In *AISTATS*, 2012.

Wang, Huayan and Koller, Daphne. Subproblem-tree calibration: A unified approach to max-product message passing. In *ICML*, 2013.

Weiss, Y. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12: 1–41, 2000.

Werner, T. A linear programming approach to max-sum problem: A review. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, July 2007.

Werner, T. High-arity interactions, polyhedral relaxations, and cutting plane algorithm for soft constraint optimisation (MAP-MRF). In *CVPR*, pp. 1–8, 2008.

Yanover, Chen, Meltzer, Talya, and Weiss, Yair. Linear programming relaxations and belief propagation - an empirical study. *Journal of Machine Learning Research*, 7:1887–1907, 2006.

Yarkony, Julian, Morshed, Ragib, Ihler, Alexander T., and Fowlkes, Charless. Tightening mrf relaxations with planar subproblems. In *UAI*, pp. 770–777, 2011.