Journal of Machine Learning Research 7 (2006) 1887–1907 Submitted 10/05; Published 9/06

Linear Programming Relaxations and Belief Propagation – An Empirical Study

Chen Yanover CHENY@CS.HUJI.AC.IL
Talya Meltzer TALYAM@CS.HUJI.AC.IL
Yair Weiss YWEISS@CS.HUJI.AC.IL
School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem, Israel

Editors: Kristin P. Bennett and Emilio Parrado-Hernández

Abstract

The problem of finding the most probable (MAP) configuration in graphical models comes up in a wide range of applications. In a general graphical model this problem is NP hard, but various approximate algorithms have been developed. Linear programming (LP) relaxations are a standard method in computer science for approximating combinatorial problems and have been used for finding the most probable assignment in small graphical models. However, applying this powerful method to real-world problems is extremely challenging due to the large numbers of variables and constraints in the linear program. Tree-Reweighted Belief Propagation is a promising recent algorithm for solving LP relaxations, but little is known about its running time on large problems.

In this paper we compare tree-reweighted belief propagation (TRBP) and powerful general-purpose LP solvers (CPLEX) on relaxations of real-world graphical models from the fields of computer vision and computational biology. We find that TRBP almost always finds the solution significantly faster than all the solvers in CPLEX and more importantly, TRBP can be applied to large scale problems for which the solvers in CPLEX cannot be applied. Using TRBP we can find the MAP configurations in a matter of minutes for a large range of real world problems.

1. Introduction

The task of finding the most probable assignment (or MAP) in a graphical model comes up in a wide range of applications including image understanding (Tappen and Freeman, 2003), error correcting codes (Feldman et al., 2003) and protein folding (Yanover and Weiss, 2002). For an arbitrary graph, this problem is known to be NP hard (Shimony, 1994) and various approximation algorithms have been proposed [see. e.g (Marinescu et al., 2003) for a recent review].

Linear Programming (LP) Relaxations are a standard method for approximating combinatorial optimization problems in computer science (Bertismas and Ttsitskikilis, 1997). They have been used for approximating the MAP problem in a general graphical model by Santos (1991). More recently, LP relaxations have been used for error-correcting codes (Feldman et al., 2003), and for protein folding (Kingsford et al., 2005). LP relaxations have an advantage over other approximate inference schemes in that they come with an optimality guarantee – when the solution to the linear program is integer, then the LP solution is guaranteed to give the global optimum of the posterior probability.

The research described in this paper grew out of our experience in using LP relaxations for problems in computer vision, computational biology and statistical physics. In all these fields, the number of variables in a realistic problem may be on the order of 10^6 or more. We found that using powerful, off-the-shelf LP solvers, these problems cannot be solved using standard desktop hardware. However, linear programs that arise out of LP relaxations for graphical models have a common structure and are a small subset of all possible linear programs. The challenge is to find an LP solver that takes advantage of this special structure.

Tree-reweighted belief propagation (TRBP) is a variant of belief propagation (BP) suggested by Wainwright and colleagues (Wainwright et al., 2002), that has been shown to find the same solution as LP relaxations. Each iteration of TRBP is similar in time and space complexity to that of ordinary BP and hence it can be straightforwardly applied to very large graphical models. However, little is known regarding the convergence properties of TRBP nor about the actual number of iterations needed to solve large problems.

In this paper we compare tree-reweighted BP and powerful commercial LP solvers (CPLEX) on relaxations of real-world graphical models from the fields of computer vision and computational biology. We find that TRBP almost always finds the solution significantly faster than all the solvers in CPLEX and more importantly, TRBP can be applied to large scale problems for which the solvers in CPLEX cannot be applied. Using TRBP we can find the MAP configurations in a matter of minutes for a large range of real world problems.

2. MAP, Integer Programming and Linear Programming

We briefly review the formalism of graphical models. We use x to denote a vector of hidden variables and y to denote the observation vector. We assume the conditional distribution $\Pr(x|y)$ is Markovian with respect to a graph G – that is, it factorizes into a product of potential functions defined on the cliques of the graph G. In this paper, we focus on pairwise Markov Random Fields and assume that

$$\begin{aligned}\Pr(x|y) &= \frac{1}{Z}\prod_{<ij>}\Psi_{ij}(x_i,x_j)\prod_i\Psi_i(x_i)\\ &= \frac{1}{Z}e^{-\sum_{<ij>}E_{ij}(x_i,x_j)-\sum_i E_i(x_i)},\end{aligned}$$

where $<ij>$ refers to all pairs of nodes that are connected in the graph G and we define $E_{ij}(x_i,x_j)$, $E_i(x_i)$ as the negative logarithm of the potential $\Psi_{ij}(x_i,x_j),\Psi_i(x_i)$.

The MAP assignment is the vector x^* which maximizes the posterior probability:

$$\begin{aligned}x^* &= \arg\max_x \prod_{<ij>}\Psi_{ij}(x_i,x_j)\prod_i\Psi_i(x_i)\\ &= \arg\min_x \sum_{<ij>}E_{ij}(x_i,x_j)+\sum_i E_i(x_i).\end{aligned}$$

To define the LP relaxation, we first reformulate the MAP problem as one of integer programming. We introduce indicator variables $q_i(x_i)$ for each individual variable and additional indicator variables $q_{ij}(x_i,x_j)$ for all connected pairs of nodes in the graph. Using these indicator variables we define the integer program:

minimize

$$J(\{q\}) = \sum_{<ij>} \sum_{x_i,x_j} q_{ij}(x_i,x_j) E_{ij}(x_i,x_j) + \sum_i \sum_{x_i} q_i(x_i) E_i(x_i)$$

subject to

$$\begin{aligned} q_{ij}(x_i,x_j) &\in \{0,1\}, \\ \sum_{x_i,x_j} q_{ij}(x_i,x_j) &= 1, \\ \sum_{x_i} q_{ij}(x_i,x_j) &= q_j(x_j), \end{aligned}$$

where the last equation enforces the consistency of the pairwise indicator variables with the singleton indicator variable.

This integer program is completely equivalent to the original MAP problem, and is hence computationally intractable. We can obtain the linear programming relaxation by allowing the indicator variables to take on non-integer values. This leads to the following problem:

The LP relaxation of MAP:
minimize

$$J(\{q\}) = \sum_{<ij>} \sum_{x_i,x_j} q_{ij}(x_i,x_j) E_{ij}(x_i,x_j) + \sum_i \sum_{x_i} q_i(x_i) E_i(x_i)$$

subject to

$$\begin{aligned} q_{ij}(x_i,x_j) &\in [0,1], && (1) \\ \sum_{x_i,x_j} q_{ij}(x_i,x_j) &= 1, && (2) \\ \sum_{x_i} q_{ij}(x_i,x_j) &= q_j(x_j). && (3) \end{aligned}$$

This is now a linear program – the cost and the constraints are linear. It can therefore be solved in polynomial time and we have the following guarantee:

Lemma If the solutions $\{q_{ij}(x_i,x_j), q_i(x_i)\}$ to the MAP LP relaxation are all *integer*, that is $q_{ij}(x_i,x_j), q_i(x_i) \in \{0,1\}$, then $x_i^* = \arg\max_{x_i} q_i(x_i)$ is the MAP assignment.+

2.1 The Need for Special Purpose LP Solvers

Given the tremendous amount of research devoted to LP solvers, it may seem that the best way to solve LP relaxations for graphical models, would be to simply use an industrial-strength, general-purpose LP solver. However, by relaxing the MAP into a linear program we increase the size of the problem tremendously. Formally, denote by k_i the number of possible states of node i. The number of variables and constraints in the LP relaxation is given by

$$\begin{aligned} N_{variables} &= \sum_i k_i + \sum_{<i,j>} k_i k_j, \\ N_{constraints} &= \sum_{<i,j>} (k_i + k_j + 1). \end{aligned}$$

The additional $\sum_{<i,j>} 2k_ik_j$ bound constraints, derived from equation (1), are usually not considered part of the constraint matrix.

As an example, consider an image processing problem (an example of such a problem, the stereo problem, is discussed in Section 4.1). If the image is a modest 200×200 pixels and each pixel can take on 30 discrete values, then the LP relaxation will have over 72 million variables and four million constraints. Obviously, we need a solver that can somehow take advantage of the problem structure in order to deal with such a large-scale problem.

3. Solving Linear Programs Using Tree-Reweighted Belief Propagation

Tree-reweighted belief propagation (TRBP) is a variant of belief propagation introduced by Wainwright and colleagues (Wainwright et al., 2002). We start by briefly reviewing ordinary max-product belief propagation [see e.g. (Yedidia et al., 2001; Pearl, 1988)]. The algorithm receives as input a graph G and the potentials Ψ_{ij}, Ψ_i. At each iteration, a node i sends a message $m_{ij}(x_j)$ to its neighbor in the graph j. The messages are updated as follows:

$$m_{ij}(x_j) \leftarrow \alpha_{ij} \max_{x_i} \Psi_{ij}(x_i, x_j)\Psi_i(x_i) \prod_{k \in N_i \setminus j} m_{ki}(x_i) \tag{4}$$

where $N_i \setminus j$ refers to all neighbors of node i except j. The constant α_{ij} is a normalization constant typically chosen so that the messages sum to one (the normalization has no influence on the final beliefs). After the messages have converged, each node can form an estimate of its local "belief" defined as

$$b_i(x_i) \propto \Psi_i(x_i) \prod_{j \in N_i} m_{ji}(x_i).$$

It is easy to show that when the graph is singly-connected, choosing an assignment that maximizes the local belief will give the MAP estimate (Pearl, 1988). In fact, when the graph is a chain, equation 4 is simply a distributed computation of dynamic programming. When the graph has cycles, ordinary BP is no longer guaranteed to converge, nor is there a guarantee that it can be used to find the MAP.

In tree-reweighted BP (TRBP), the algorithm receives as input an additional set of *edge appearance probabilities*, ρ_{ij}. These edge appearance probabilities are essentially free parameters of the algorithm and are derived from a distribution over spanning trees of the graph G. They represent the probability of an edge (ij) appearing in a spanning tree under the chosen distribution. As in standard belief propagation, at each iteration a node i sends a message $m_{ij}(x_j)$ to its neighbor in the graph j. The messages are updated as follows:

$$m_{ij}(x_j) \leftarrow \alpha_{ij} \max_{x_i} \Psi_{ij}^{1/\rho_{ij}}(x_i, x_j)\Psi_i(x_i) \frac{\prod_{k \in N_i \setminus j} m_{ki}^{\rho_{ki}}(x_i)}{m_{ji}^{1-\rho_{ji}}(x_i)}. \tag{5}$$

Note that for $\rho_{ij} = 1$ the algorithm reduces to standard belief propagation.

After one has found a fixed-point of these message update equations, the singleton and pairwise beliefs are defined as

$$b_i(x_i) \quad \propto \quad \Psi_i(x_i) \prod_{j \in N_i} m_{ji}^{\rho_{ji}}(x_i),$$

$$b_{ij}(x_i,x_j) \quad \propto \quad \Psi_i(x_i)\Psi_j(x_j)\Psi_{ij}^{1/\rho_{ij}}(x_i,x_j)\cdot\frac{\prod_{k\in N_i\setminus j} m_{ki}^{\rho_{ki}}(x_i)}{m_{ji}^{1-\rho_{ji}}(x_i)}\frac{\prod_{k\in N_j\setminus i} m_{kj}^{\rho_{kj}}(x_j)}{m_{ij}^{1-\rho_{ij}}(x_j)}.$$

The relationship between TRBP and the solution to the LP relaxation has been studied by (Wainwright et al., 2002; Kolmogorov, 2005; Kolmogorov and Wainwright, 2005) and is a subject of ongoing research. We briefly summarize some of the relationships.

Given a set of TRBP beliefs, we define the *sharpened beliefs* as follows:

$$\begin{aligned} q_i(x_i) &\propto \delta(b_i(x_i)-\max_{x_i} b_i(x_i)), \\ q_{ij}(x_i,x_j) &\propto \delta(b_{ij}(x_i,x_j)-\max_{x_i,x_j} b_{ij}(x_i,x_j)), \end{aligned}$$

where $\delta(\cdot)$ is the Dirac delta function ($\delta(0)=1$ and $\delta(x)=0$ for all $x\neq 0$). That is, we get a uniform distribution over all the maximizing values and assign 0 probability to all non-maximizing values. To illustrate this definition, a belief vector $(0.6,0.4)$ would be sharpened to $(1,0)$ and a belief vector $(0.4,0.4,0.2)$ would be sharpened to $(0.5,0.5,0)$.

Using these sharpened beliefs, the following properties hold:

- At any iteration, and in particular in fixed-point, the TRBP beliefs provide a lower bound on the solution of the LP (see appendix A).
- If there exists a unique maximizing value for the pairwise beliefs $b_{ij}(x_i,x_j)$ then the sharpened beliefs solve the LP. In that case $x_i^*=\arg\max_{x_i} b_i(x_i)$ is the MAP.
- Suppose the TRBP beliefs have ties. If there exists x^* such that x_i^*,x_j^* maximize $b_{ij}(x_i,x_j)$ and x_i^* maximize $b_i(x_i)$, then x^* is the MAP. In that case, define q_{ij}^*,q_i^* as indicator variables for x^*, then q_{ij}^*,q_i^* are a solution for the LP.
- If the sharpened beliefs at a fixed-point of TRBP satisfy the LP constraints (equations 1-3), then the sharpened beliefs are a solution to the LP relaxation.
- Suppose the TRBP beliefs have ties. If there exists $\tilde{b}_i,\tilde{b}_{ij}$ that satisfy the LP constraints and for all x_i,x_j, $\tilde{b}_i(x_i)=0$ if $q_i(x_i)=0$ and $\tilde{b}_{ij}(x_i,x_j)=0$ if $q_{ij}(x_i,x_j)=0$, then $\tilde{b}_i,\tilde{b}_{ij}$ are a solution to the LP relaxation.

The lower-bound property is based on Lagrangian duality and is proven in (Wainwright et al., 2002; Kolmogorov, 2005). The subsequent properties follow from the lower-bound property.

Based on these properties, we use the following algorithm to extract the LP solution and the MAP from TRBP beliefs:

1. Run TRBP until convergence and identify the tied nodes x_T.
2. For all non-tied nodes, x_{NT}, set $x_i^*=\arg\max_{x_i} b_i(x_i)$.
3. Construct a new graphical model that includes only the tied nodes and the possible states are only those that maximize the beliefs. The pairwise potentials are 1 if the pair maximizes the pairwise belief and ε otherwise. Use the junction tree algorithm (Cowell, 1998) to find x_T^*, the MAP in this new graphical model. If x_T^* has energy equal to zero then $x^*=(x_{NT}^*,x_T^*)$ is the MAP and q_{ij}^*,q_i^*, defined as indicator variables for x^*, are a solution to the LP.

3.1 TRBP Complexity

Little is known about the number of iterations needed for TRBP to converge, but what is relatively straightforward to calculate is the cost per iteration. In every TRBP iteration, each node calculates and sends messages to all its neighbors. Thus the number of message updates is two times the number of edges in the graph. Each message update equation involves point-wise multiplication of vectors of length k_i followed by a matrix by vector max-multiplication,[1] where the size of the matrix is $k_j \times k_i$. By working in the log domain, the point-wise multiplications can be transformed into summations, and the raising of the messages to the powers $\rho_{ij}, 1-\rho_{ij}$ are transformed into multiplications. In summary, the dominant part of the computation is equivalent to $2|\mathcal{E}|$ multiplications of a $k_i \times k_j$ matrix times a $k_j \times 1$ vector (where $|\mathcal{E}|$ is the number of edges in the graph).

The amount of memory needed depends on the particular implementation. In the simplest implementation, we would need to store the potentials Ψ_{ij}, Ψ_i in memory. The size of the potentials is exactly the number of variables in the linear program: $\sum_i k_i + \sum_{<i,j>} k_i k_j$. Additionally, storing the messages in memory requires $\sum_{<i,j>} k_i + k_j$ (which is typically small relative to the memory required for the pairwise potentials). In many problems, however, the pairwise potentials can be stored more compactly. For example, in the Potts model, the pairwise potential $\Psi_{ij}(x_i, x_j)$ is a $k \times k$ table that has only two unique values: 1 on the diagonal and $e^{-\lambda_{ij}}$ for the off-diagonal terms. Additional implementation techniques for reducing the memory requirements of BP appear in (Felzenszwalb and Huttenlocher, 2004; Kolmogorov, 2005).

4. The Benchmark Problems

We constructed benchmark problems from three domains: stereo vision, side-chain prediction and protein design. We give here a short overview of how we constructed the graphical models in all three cases. The exact graphical models can be downloaded from the JMLR web site.

4.1 Stereo Vision

The stereo problem is illustrated in Figure 1. Given a stereo pair of images, $\mathit{Left}(u,v)$ and $\mathit{Right}(u,v)$, the problem is to find the disparity of each pixel in a reference image. This disparity can be straightforwardly translated into depth from the camera.

The main cue for choosing disparities are the similarities of local image information in the left and right image. That is, we search for a disparity so that

$$\mathit{Left}(u,v) \approx \mathit{Right}(u + disp(u,v), v),$$

or equivalently,

$$disp^*(u,v) = \arg \min_{disp(u,v)} \left[\mathit{Left}(u,v) - \mathit{Right}(u + disp(u,v), v)\right]^2.$$

This criterion by itself is typically not enough. Locally, there can be many disparities for a pixel that are almost equally good. The best algorithms currently known for the stereo problem are those that minimize a global energy function (Scharstein and Szeliski, 2002):

$$disp^* = \arg\min_{disp} \sum_{u,v} dissim\left[\mathit{Left}(u,v), \mathit{Right}(u + disp(u,v), v)\right] + \lambda \cdot smoothness(disp),$$

1. Max-multiplication of a matrix by a vector is equivalent to ordinary matrix multiplication but all summations are replaced by maximizations.

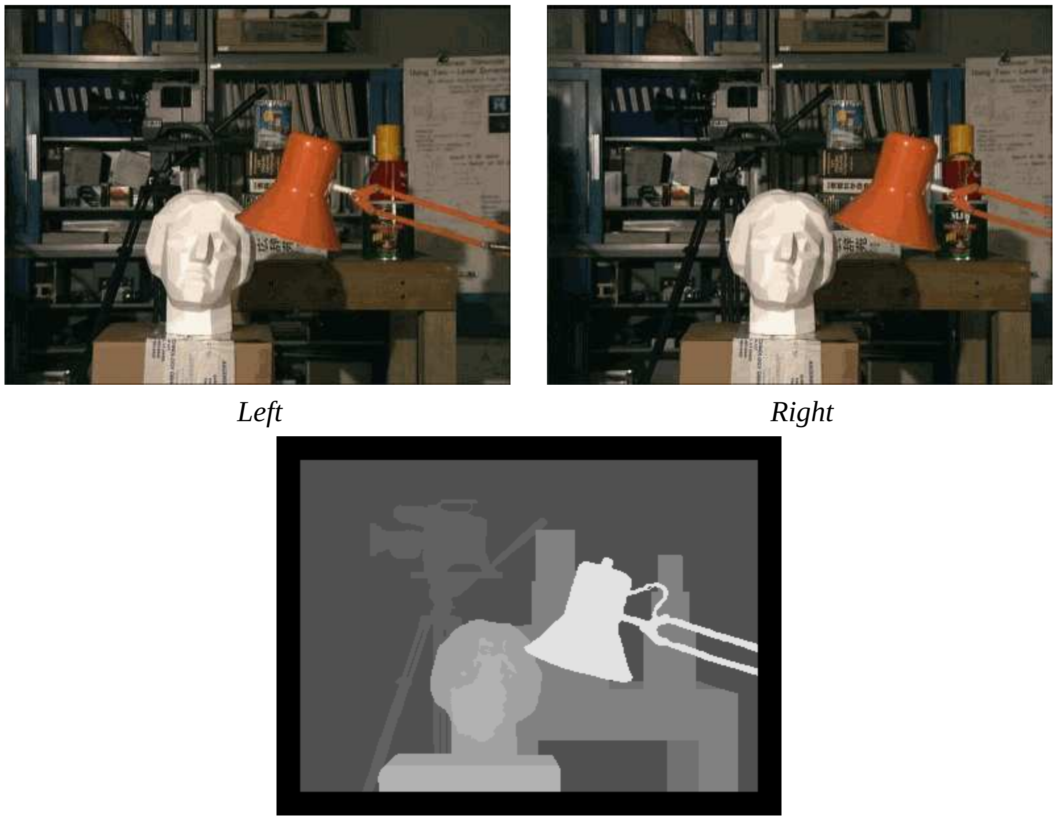

Figure 1: An illustration of the stereo problem. Given two images taken from slightly different viewpoints (*Left*, *Right*) we search for the disparity of each pixel. The best results for this problem use energy minimization formulations which are equivalent to solving the MAP for a grid graphical model.

where $smoothness(disp)$ is a cost that penalizes disparity fields where neighboring pixels have different disparities and $dissim[Left(u,v), Right(u',v')]$ measures the dissimilarity of the left and right image at corresponding locations.

We associate each disparity $disp(u,v)$ with an assignment of a node x_i in a two dimensional grid graph. If we define x to be the disparity field, and $P(x|y) \propto \exp(-E(x))$ where $E(x)$ is the energy function, minimizing the energy is equivalent to maximizing P(x). Furthermore, since $E(x)$ is a sum of singleton and pairwise terms, $P(x)$ will factorize with respect to the two-dimensional grid:

$$\begin{aligned} Pr(x|y) &\propto \prod_i \Psi_i(x_i) \prod_{<ij>} \Psi_{ij}(x_i,x_j) = \\ &= e^{-\sum_i E_i(x_i) - \sum_{<ij>} E_{ij}(x_i,x_j)}. \end{aligned}$$

The problem of finding the most probable set of disparities is NP hard. Good approximate solutions can be achieved using algorithms based on min-cut/max-flow formulations (Boykov et al.,

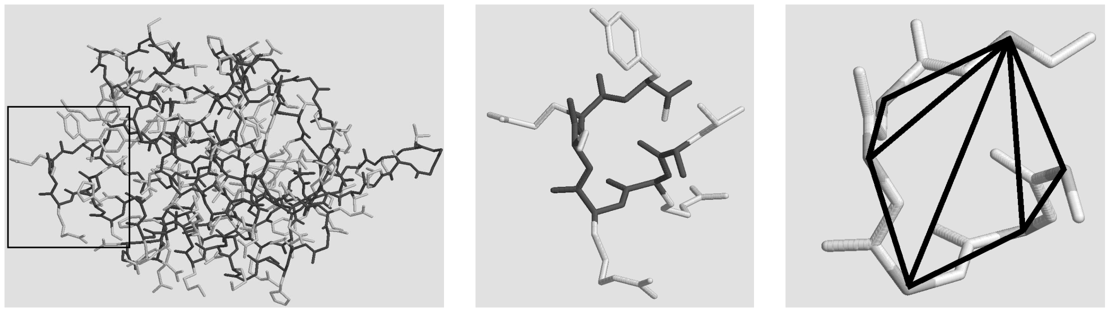
(a) (b) (c)

Figure 2: **(a)** Cow actin binding protein (PDB code 1pne). **(b)** A closer view of its 6 C-terminal residues. Given the protein backbone (black) and the amino acid sequence, side-chain prediction is the problem of predicting the native side-chain conformation (gray). **(c)** Problem representation as a graphical model for those C-terminal residues shown in (b) (nodes located at C^α atom positions, edges drawn in black).

1999; Kolmogorov and Zabih, 2004) and Belief Propagation (Felzenszwalb and Huttenlocher, 2004; Tappen and Freeman, 2003; Sun et al., 2002).

In this work we use the same energy function used by Tappen and Freeman (2003). The local cost is based on the Birchfield-Tomasi matching cost (Birchfield and Tomasi, 1998) and the pairwise energy penalizes for neighboring pixels having different disparities. The amount of penalty depends only on the intensity difference between the two pixels and therefore, for each pair of neighboring pixels, the penalty for violating the smoothness constraint is constant. Thus the MRF is equivalent to a Potts model. Specifically, the pairwise energy penalty is defined using 3 parameters – s, P and T – and set to $P \cdot s$ when the intensity difference between the two pixels is smaller than a threshold T, and s otherwise.

We used four images from the standard Middlebury stereo benchmark set (Scharstein and Szeliski, 2003). By varying the parameters of the energy function, as in (Tappen and Freeman, 2003), we obtained 22 different graphical models. The parameters s, P, T are constant over the whole image.

4.2 Side-Chain Prediction

Proteins are chains of simpler molecules called *amino acids*. All amino acids have a common structure – a central carbon atom (C^α) to which a hydrogen atom, an amino group (NH_2) and a carboxyl group ($COOH$) are bonded. In addition, each amino acid has a chemical group called the *side-chain*, bound to C^α. This group distinguishes one amino acid from another and gives its distinctive properties. Amino acids are joined end to end during protein synthesis by the formation of peptide bonds. An amino acid unit in a protein is called a *residue*. The formation of a succession of peptide bonds generates the *backbone* (consisting of C^α and its adjacent atoms, N and CO, of each reside), upon which the side-chains are hanged (Figure 2).

The *side-chain prediction* problem is defined as follows: given the 3 dimensional structure of the backbone we wish to predict the placements of the side-chains. This problem is considered of central importance in protein-folding and molecular design and has been tackled extensively using a wide variety of methods. Typically, an energy function is defined over a discretization of the side-chain angles and search algorithms are used to find the global minimum. Even when the energy function contains only pairwise interactions, the configuration space grows exponentially and it can be shown that the prediction problem is NP-complete (Fraenkel, 1997; Pierce and Winfree, 2002).

Formally, our search space is a set of energetically preferred conformations (called *rotamers*) and we wish to minimize an energy function that is typically defined in terms of pairwise interactions among nearby residues and interactions between a residue and the backbone:

$$E(r) = \sum_{<ij>} E_{ij}(r_i, r_j) + \sum_i E_i(r_i, backbone),$$

where $r = (r_1, ..., r_N)$ denotes an assignment of rotamers for all residues.

Since we have a discrete optimization problem and the energy function is a sum of pairwise interactions, we can transform the problem into a graphical model with pairwise potentials. Each node corresponds to a residue, and the state of each node represents the configuration of the side-chain of that residue. Since $E(r)$ is a sum of singleton and pairwise terms, $P(r)$ will factorize:

$$\begin{aligned} P(r) = \frac{1}{Z}e^{-E(r)} &= \frac{1}{Z}e^{-\sum_i E_i(r_i) - \sum_{<ij>} E_{ij}(r_i,r_j)} \\ &= \frac{1}{Z}\prod_i \Psi_i(r_i) \prod_{<ij>} \Psi_{ij}(r_i, r_j) \end{aligned} \tag{6}$$

where Z is an explicit normalization factor. Equation (6) requires multiplying Ψ_{ij} for all pairs of residues i, j but in all reasonable energy functions the pairwise interactions go to zero for atoms that are sufficiently far away. Thus we only need to calculate the pairwise interactions for nearby residues. To define the topology of the undirected graph, we examine all pairs of residues i, j and check whether there exists an assignment r_i, r_j for which the energy is nonzero. If it exists, we connect nodes i and j in the graph and set the potential to be: $\Psi_{ij}(r_i, r_j) = e^{-E_{ij}(r_i,r_j)}$.

Figure 2(c) shows a subgraph of the undirected graph. The graph is relatively sparse (each node is connected to nodes that are close in 3D space) but contains many small loops. A typical protein in the data set gives rise to a model with hundreds of loops of size 3.

As a data set we used 370 X-ray crystal structures with resolution better than or equal to 2Å, R factor below 20% and mutual sequence identity less than 50%. Each protein consisted of a single chain and up to 1,000 residues. Protein structures were acquired from the Protein Data Bank site (`http://www.rcsb.org/pdb`). For each protein, we have built two representing graphical models:

1. Using the SCWRL energy function (Canutescu et al., 2003), which approximates the repulsive portion of Lennard-Jones 12-6 potential.

2. Using the more elaborate energy function used in the Rosetta program (Kuhlman and Baker, 2000) which is comprised of (Rohl et al., 2004): (1) the attractive portion of the 12-6 Lennard-Jones potential, (2) The repulsive portion of a 12-6 Lennard Jones potential. This term is dampened in order to compensate for the use of a fixed backbone and rotamer set, (3) solvation energies calculated using the model of Lazaridis and Karplus (1999), (4) an approximation to electrostatic interactions in proteins, based on PDB statistics, (5) hydrogen-bonding potential (Kortemme et al., 2003) and (6) backbone dependent internal free energies of the rotamers estimated from PDB statistics performed by Dunbrack and Kurplus (1993).

The Rosetta energy function accounts for distant interactions and therefore gives rise to denser graphical models, compared to SCWRL's. In both cases we used Dunbrack and Kurplus (1993) backbone dependent rotamer library to define up-to 81 configurations for each side-chain. The side-chain prediction data sets are publicly available and can be downloaded from `http://www.jmlr.org/papers/volume7/yanover06a/SCWRL_SCP_Dataset.tgz` and `http://www.jmlr.org/papers/volume7/yanover06a/Rosetta_SCP_Dataset.tgz`.[2]

4.3 Protein Design

The protein design problem is the inverse of the protein folding problem. Given a particular 3D shape, we wish to find a sequence of amino-acids that will be as stable as possible in that 3D shape. Typically this is done by finding a set of (1) amino-acids and (2) rotamer configurations that minimizes an approximate energy [see (Street and Mayo, 1999) for a review of computational protein design].

While the problem is quite different from side-chain prediction it can be solved using the same graph structure. The only difference is that now the nodes do not just denote rotamers but also the identity of the amino-acid at that location. Thus, the state-space here is significantly larger than in the side-chain prediction problem. We, again, used the Rosetta energy function to define the pairwise and local potentials (Kuhlman and Baker, 2000). As a data set we used 97 X-ray crystal structures, 40-180 amino acids long. For each of these proteins, we allowed all residues to assume any rotamer of any amino acid. There are, therefore, hundreds of possible states for each node. The protein design data set is available from `http://www.jmlr.org/papers/volume7/yanover06a/Rosetta_Design_Dataset.tgz`.

5. Experiments

We compared TRBP to the LP solvers available from CPLEX (CPLEX *v*9.0, `tomlab.biz`). CPLEX is widely considered to be one of the most powerful LP packages available commercially and (according to the company's website) is used in 95% of all academic papers that reference an LP solver. In addition to its widespread use, CPLEX is well suited for our empirical study because it is highly optimized for solving LP relaxations (as opposed to arbitrary LPs). Indeed as the original programmer of CPLEX notes "the solution of integer programs is the dominant application of linear programming in practice" (Bixby, 2001) and CPLEX contains a large number of optimizations that are based on exploiting the sparse structure of LPs that arise from LP relaxations (Bixby, 2001).

Specifically, we tried the following solvers from CPLEX.

1. *Primal* and *dual* simplex solvers.

2. CPLEX has a very efficient algorithm for *network* models. Network constraints have the following property: each non-zero coefficient is either a $+1$ or a -1 and column appearing in these constraints has exactly 2 nonzero entries, one with a $+1$ coefficient and one with a -1 coefficient. CPLEX can also automatically extract networks that do not adhere to the above conventions as long as they can be transformed to have those properties.

2. Note that loading a protein file in Matlab requires using the sparse_cell package available from `http://www.cs.huji.ac.il/~cheny`.

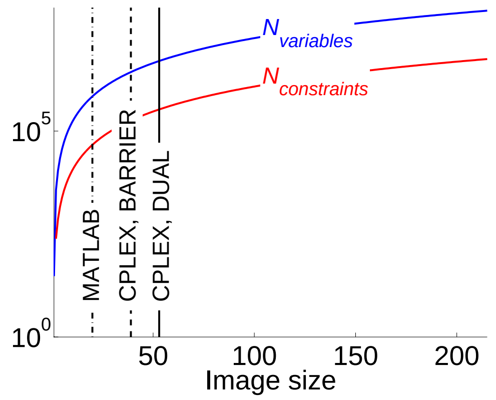

Figure 3: The number of variables and constraints in the LP relaxation of the stereo disparity problem as a function of the size of the image. The largest image that could be solved using the CPLEX solvers is approximately 50×50 while TRBP can be run on full size images.

3. The *barrier* algorithm is a primal-dual logarithmic barrier algorithm which generates a sequence of strictly positive primal and dual solutions. The barrier algorithm is highly optimized for large, sparse problems.

4. CPLEX provides a *sifting* algorithm which can be effective on problems with many more variables than equations. Sifting solves a sequence of LP subproblems where the results from one subproblem are used to select columns from the original model for inclusion in the next subproblem.

5. The *concurrent* optimizer can apply multiple algorithms to a single linear programming problem. Each algorithm operates on a different CPU.

We used the `tomlab` package that provides a Matlab interface to CPLEX 9.0. Our TRBP implementation was written in C++ and linked to Matlab as a cmex file. The TRBP implementation is completely general and receives as input a graph and the potential functions. It iterates the message updating equations (equation 5) until convergence. To improve the convergence properties "dampening" is used – we only move the new messages halfway towards the new value of the message. The messages are represented in the log domain so that multiplication is replaced with summation. Convergence is declared when the beliefs change by no more than 10^{-8} between successive iterations and the same threshold is used to determine ties. The edge appearance probabilities ρ_{ij} are automatically calculated for a given graph by greedily constructing a set of spanning trees until all edges in the graph appear in exactly one spanning tree. The junction tree algorithm needed for the post-processing of the TRBP beliefs is performed in Matlab using Kevin Murphy's BNT package (Murphy, 2001). Despite the Matlab implementation, the junction tree run-time is negligible compared to the TRBP run times (typically less than 30 seconds for the junction tree). We also compared the run-times of ordinary BP by running the same code but with $\rho_{ij} = 1$ for all edges. All algorithms were run on a dual processor Pentium 4 with 4G memory (but using a single processor only).

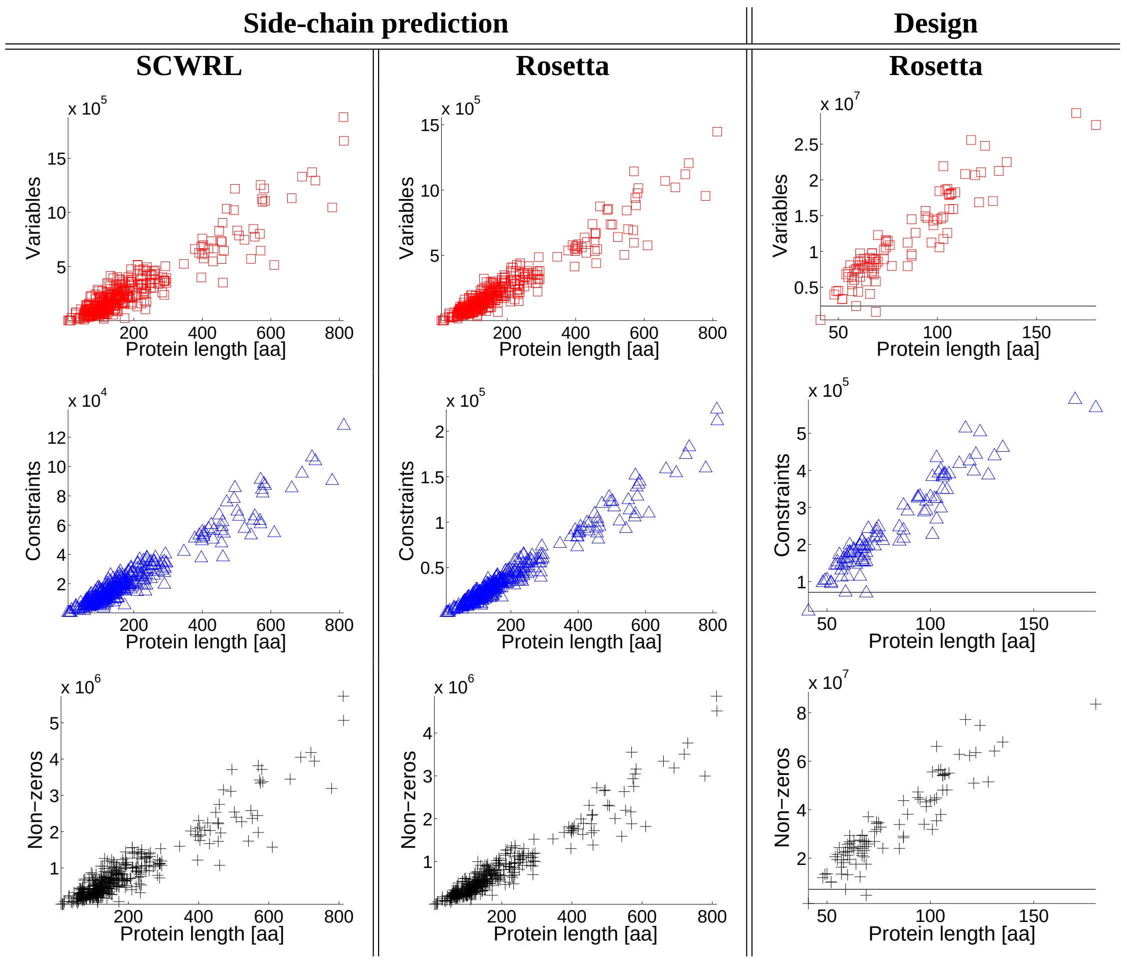

Figure 4: The number of variables, constraints and non-zero entries in the constraint matrix for our benchmark problems in side-chain prediction, using SCWRL (left) and Rosetta (middle) energy functions, and protein design (right). For the side-chain prediction problem both TRBP and the CPLEX solvers could solve the LP relaxation for all proteins in the database. For the protein design problem, on the other hand, the CPLEX solvers could only solve a small fraction of the database (3/97) while TRBP could solve the relaxations for all the proteins in the database (the horizontal line in the plots in the right column indicates the largest model that could be solved using CPLEX).

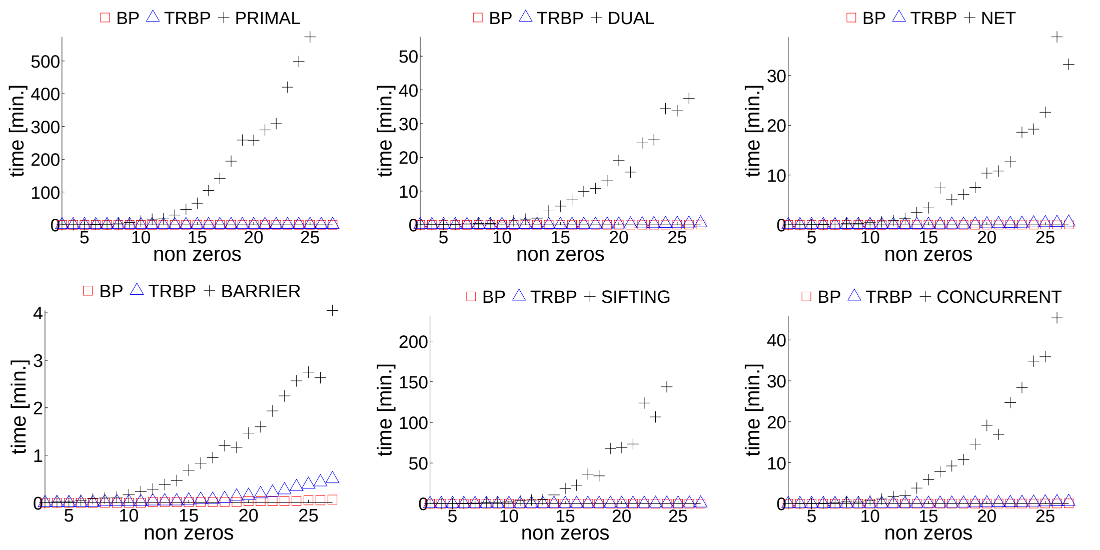

Figure 5: Comparison of run-times of the six solvers from CPLEX and TRBP on a set of subproblems constructed from the “map” image in the Middlebury stereo benchmark set. The barrier method is the fastest of the CPLEX solvers but it is still significantly slower than TRBP for relatively large problems.

The first question we asked was: what is the largest problem in each data set that can be solved within 12 hours by each of the solvers ? Figure 3 shows the results for a standard stereo benchmark image (the “map” image from the Middlebury stereo benchmark set (Scharstein and Szeliski, 2003)). We constructed smaller problems by taking subimages from the full image. Out of the CPLEX solvers, the dual simplex algorithm could solve the largest subproblem (the barrier algorithm requires more memory) but it could not solve an image larger than approximately 50×50 pixels. In contrast, TRBP can be run on the full benchmark images (approximately 250×250 pixels). Figure 4 shows the problem sizes for the side-chain prediction and the protein design problems. For the side-chain prediction problem all solvers could be applied to the full benchmark set. However for the protein design problem (in which the state space is much larger) the CPLEX solvers could solve only 2 out of the 96 problems in the database (this is indicated by the horizontal line in the plots in the right column) while TRBP could solve them all.

In the second experiment we asked: how do the run-times of the solvers compare in settings where all solvers can be applied. Figure 5 compares the run-times on the sequence of subproblems constructed from the Middlebury stereo benchmark set. As can be seen, the barrier method is the fastest of the CPLEX solvers but it is still significantly slower than TRBP on large problems.

Figure 6 compares the run times of the different solvers on the side-chain prediction graphical models. Again, the barrier method is the fastest of the CPLEX solvers (with dual simplex and network solvers providing similar performance with less memory requirements) but is significantly slower than TRBP for large problems. Figure 7 shows the run times of TRBP, BP, and the barrier CPLEX solver on the protein design problem. For the few cases for which the barrier method did not run out of memory, TRBP is significantly faster.

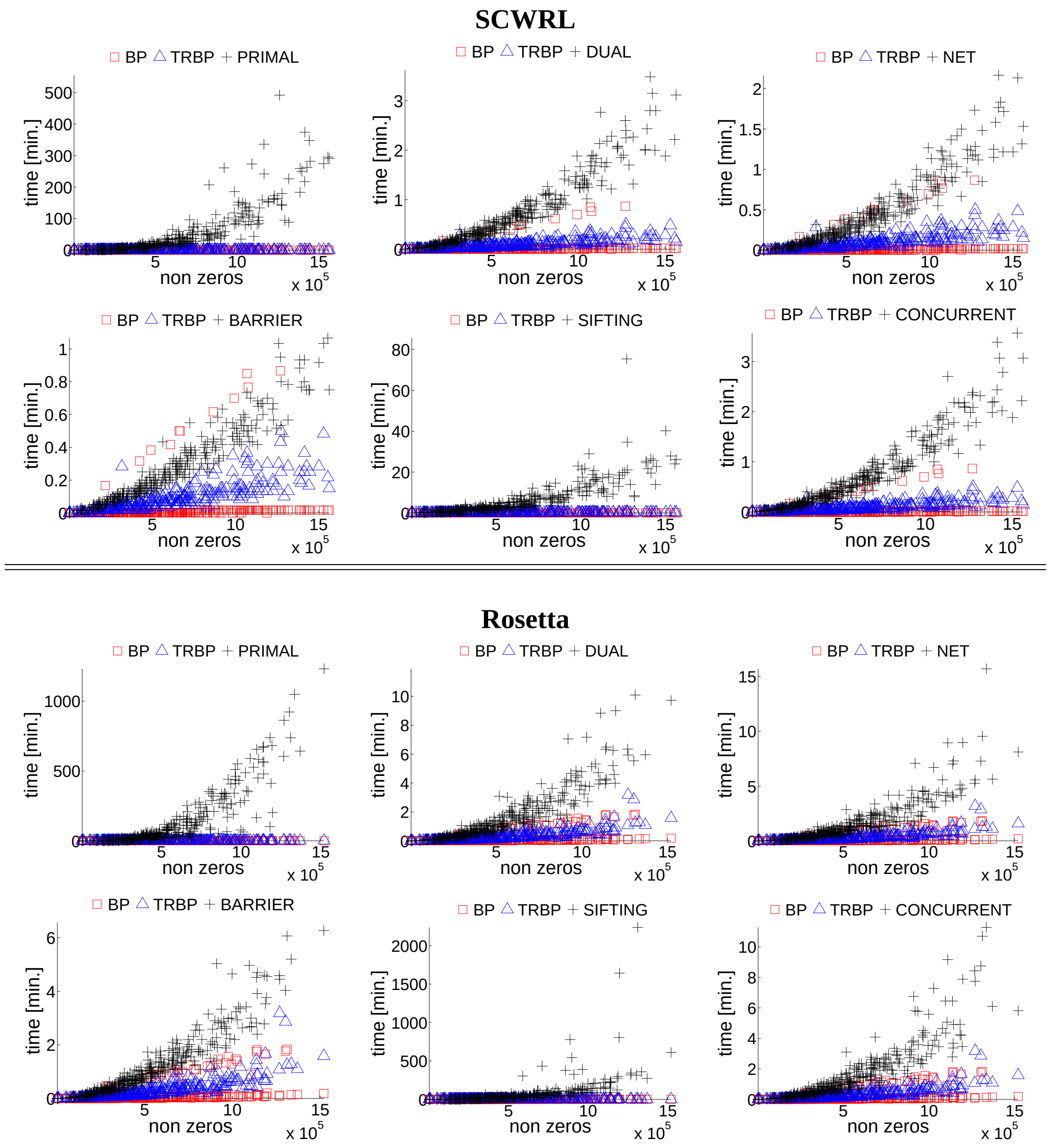

Figure 6: A comparison of the run-times of the different solvers in CPLEX and TRBP on the side-chain prediction benchmark. Again the barrier method is the fastest of the CPLEX solvers (with dual simplex and the network solver providing similar performance with less memory requirements). TRBP consistently converges faster than the barrier method and the difference becomes more significant as the problem size increases.

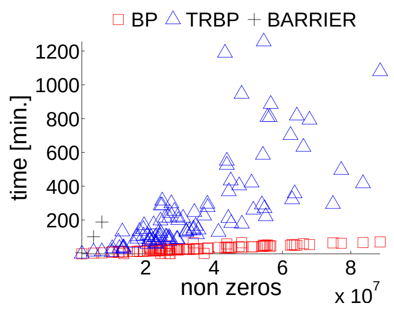

Figure 7: A comparison of the run-times of the barrier method and TRBP on the protein design problem. For the few cases in which the barrier method did not run out of memory, TRBP is significantly faster.

In the third set of experiments we asked: in what fraction of the runs can we use the results of the LP relaxation to find the MAP? We define a run of TRBP as "successful" if the TRBP beliefs allowed us to find the MAP of the graphical model (i.e. if we could find an assignment x^* that maximized the pairwise and singleton beliefs). In the stereo benchmark we could directly find the MAP in 12 out of 22 cases, but by using additional algorithms on the TRBP output we could find the MAP on 19 out of the 22 cases (Meltzer et al., 2005). Figure 8 shows the success rate for TRBP in the side-chain prediction problems. For these problems, TRBP's success rate was over 90% for proteins in our database with length less than 200 amino acids. As the proteins become larger, the problem becomes more complex and the success rate decreases. The figures also show the fraction of times in which the TRBP beliefs allowed us to solve the linear program. The protein design problem is, apparently, a more difficult problem and the success rate is therefore much lower – the MAP assignment could be found for 2 proteins only and TRBP beliefs allowed us to solve the LP relaxations for 6 proteins only. Note, however, that we could still use the TRBP beliefs to obtain a lower bound on the optimal solution.

We also assessed the success rate of the standard LP solvers, which we defined as a case when the LP solution was nonfractional. We found that *in all cases in which the LP solution was nonfractional the TRBP beliefs had a unique maximum*. Thus the success rate of the standard LP solvers was strictly less than that of TRBP (since TRBP also allows for obtaining a solution with partially tied beliefs).

6. What is TRBP's Secret?

Given the performance advantages of TRBP over the solvers in CPLEX, it is natural to ask "what is TRBP's secret?". The first thing to emphasize in this context is that *TRBP is not a general purpose LP solver*. It can only solve a tiny fraction of linear programs with a very special structure.

To see this structure, consider the general LP problem: minimize $c^T q$ subject to $Aq = b$ and $Cq < d$. If we translate LP relaxations of MAP into this form we find that the equality matrix A, the inequality matrix C, and the vectors b, d all contain only elements in $\{-1, 0, 1\}$. Tardos (1986) has shown that linear programs with integer constraint matrices can be solved with a strongly

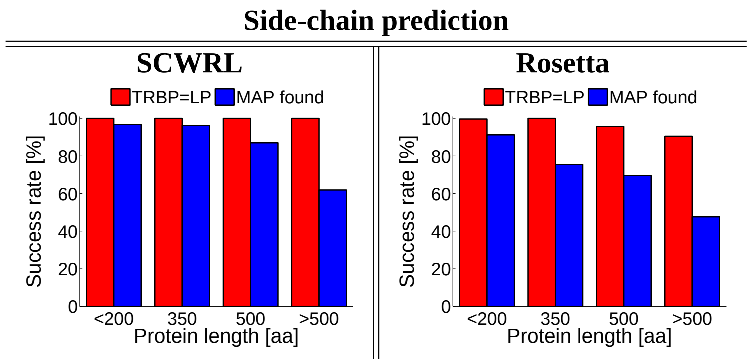

Figure 8: Success rate of TRBP on the side-chain prediction problems. A run of the algorithm was considered a success if we could use the TRBP beliefs to find the MAP of the graphical model. The figures also show the fraction of times in which the TRBP beliefs allowed us to solve the linear program.

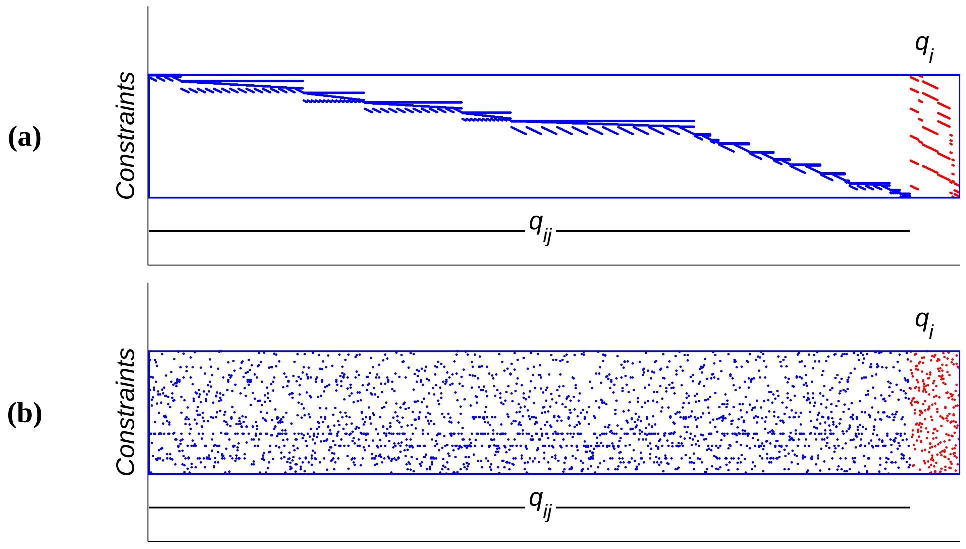

Figure 9: The sparsity pattern of a typical equality matrix A **(a)** and a random permutation of this matrix **(b)**. Blue and red dots indicate $+1$ and -1 entries respectively.

polynomial algorithm (suggesting that they are easier to solve than general purpose LPs for which no strongly polynomial algorithm is known).

The matrix A that arises in LP relaxations of MAP has additional structure, beyond the fact that its elements are in $\{-1,0,1\}$. Figure 9(a) shows the sparsity pattern of the matrix A for a small graphical model. The matrix is sparse and has a special block form. The special structure arises from the fact that we only have a consistency constraint for a pairwise indicator q_{ij} to the two singleton indicators, that involve nodes i and j. There is no interaction between the pairwise indicators q_{ij} and any other pairwise indicator q_{kl} nor is there an interaction with any other singleton

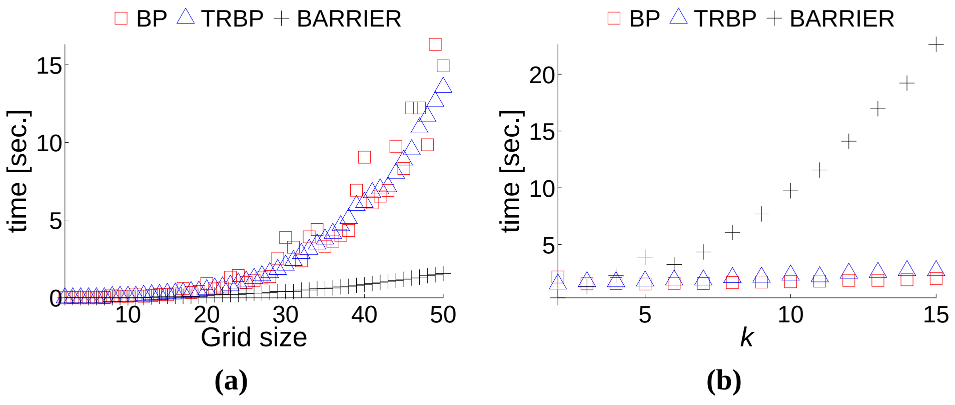

Figure 10: A comparison of the run-times of the barrier method and TRBP on **(a)** binary spin glass models (Potts models with positive and negative λ_{ij}), as a function of grid size and on **(b)** a 25×25 grid Potts model, as a function of the number of possible states, k. Each datapoint represents the average over 10 random samplings of λ_{ij}.

indicator q_k in the graph. For comparison, Figure 9(b) shows a random permutation of the matrix – this has the same sparsity pattern but without the block structure.

Note that TRBP does not even represent the matrix A explicitly. Instead, TRBP explicitly represents the graph G which implicitly defines the matrix A. In contrast, the CPLEX solvers explicitly represent A and this matrix implicitly represents the graph G (by finding the correct permutation of A that reveals the block structure, it is possible to reconstruct the graph G). We believe that this difference in representation may be responsible for TRBP's superior performance.

To investigate the conjecture that TRBP's advantage is related to an explicit representation of the graph structure, we compared the run-times of the barrier LP solver and TRBP on spin glass models (Potts models with positive and negative λ_{ij}) with different numbers of possible states per node, k. Note that the size of the block in the constraint matrix in Figure 9 is directly related to k – for binary nodes the blocks are of size 5×4 and we conjectured that when the blocks are small, TRBP's advantage will decrease.

As Figure 10(a) shows, for binary nodes the barrier solver was consistently faster than TRBP. However, as we increased k, and consequently – the size of the blocks in A, the barrier solver became much slower than TRBP (Figure 10(b)). This seems to support the assumption that the explicit representation of block structure was responsible for TRBP's superior performance in our benchmark set. In the benchmark set, k was (at least) in the order of dozens. We should also note, that even for binary problems, the barrier method will run out of memory much faster than TRBP.

7. Discussion

As pointed out in (Bixby, 2001), advances in hardware and in LP algorithms have greatly expanded the size of problems that can be solved using LP relaxations. Despite this progress, many real world problems are still too large to be handled using desktop hardware and standard LP solvers. In this paper we have experimented with the powerful solvers in CPLEX on LP relaxations of the MAP problem for graphical models from the fields of computer vision and computational biology. Despite the many optimizations in CPLEX for exploiting sparsity, we found that many of the graphical models gave rise to linear programs that were beyond the capability of all the solvers in CPLEX. In contrast, tree-reweighted BP could be applied to all the linear programs in our database and almost always gave faster solutions. By running the junction tree algorithm on a reduced graphical model defined by the nodes for which the TRBP beliefs had ties, we could find the MAP solution for a large range of real-world problems.

The LP solvers available in CPLEX are of course only a subset of the large number of LP algorithms suggested in the literature and it may very well be possible to design LP solvers that outperform TRBP on our benchmark set. To stimulate research in this direction, both the linear programs used in this paper and our implementation of TRBP are available on the internet. One direction of research that we are currently working on, involves tighter LP relaxations (Meltzer et al., 2005). As the problems become more complex, the standard LP relaxation of MAP is apparently not tight enough and solving the LP often does not enable solving for the MAP. We are exploring methods for solving a sequence of tighter and tighter relaxations and are interested in a method that will allow us to use some of the computations used in one relaxation in solving a tighter relaxation. We believe this research direction offers great potential benefit for interaction between researchers in the field of graphical models and convex optimization.

Acknowledgments

C. Y. is supported by Yeshaya Horowitz Association through the Center for Complexity Science.

Appendix A. Deriving Bounds for the LP Solution Using TRBP

In this section, we give the formula for calculating a bound on the LP solution from TRBP fixed-point beliefs b_{ij}, b_i. We assume that the beliefs have been normalized so that $\max_{x_i,x_j} b_{ij}(x_i,x_j) = 1$ and $\max_{x_i} b_i(x_i) = 1$. Note that this normalization does not change the nature of fixed-points so in case we have any set of fixed-point beliefs, we can just divide every pairwise belief by the maximal value in that belief and similarly divide every singleton belief by its maximal value. The normalized beliefs will still be fixed-points.

It can be shown (Wainwright et al., 2002) that any fixed-point of TRBP satisfies the "admissibility" equation. For any assignment x, the probability (or equivalently the energy) can be calculated from the original potentials or from the beliefs:

$$\begin{aligned} Z\,\Pr(x) &= \prod_{ij} \Psi_{ij}(x_i,x_j)\Psi_i(x_i) \\ &= K(b) \prod_{ij} b_{ij}^{\rho_{ij}}(x_i,x_j) \prod_i b_i^{c_i}(x_i) \end{aligned}$$

with $c_i = 1 - \sum_j \rho_{ij}$ and $K(b)$ is a constant *independent of* x. $K(b)$ can be calculated from any assignment x, e.g. $x^0 = 0$ where all nodes are in their first state, by

$$K(b) = \frac{\prod_{ij} \Psi_{ij}(x_i^0, x_j^0)\Psi_i(x_i^0)}{\prod_{ij} b_{ij}^{\rho_{ij}}(x_i^0, x_j^0) \prod_i b_i^{c_i}(x_i^0)} \tag{7}$$

Similarly, it can be shown that for any q_{ij}, q_i that satisfy the LP constraints, one can calculate the energy from the beliefs:

$$\begin{aligned} J(q) &= \sum_{<ij>} \sum_{x_i,x_j} q_{ij}(x_i,x_j) E_{ij}(x_i,x_j) + \sum_i \sum_{x_i} q_i(x_i) E_i(x_i) \\ &= -\ln K(b) - \sum_{<ij>} \rho_{ij} \sum_{x_i,x_j} q_{ij}(x_i,x_j) \ln b_{ij}(x_i,x_j) - \sum_i c_i \sum_{x_i} q_i(x_i) \ln b_i(x_i). \end{aligned}$$

By using the admissibility constraint and the properties of the numbers ρ_{ij}, c_i it can be shown that $J(q) \geq -\ln K(b)$. Direct inspection shows that if $q_i j, q_i$ are the sharpened beliefs then they achieve the bound (since they are nonzero only when $b_{ij}(x_i,x_j) = 1$ or $b_i(x_i) = 1$).

References

D. Bertismas and J. Ttsitskikilis. *Introduction to linear optimization*. Athena Scientific, 1997.

S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):401–406, 1998.

R. E. Bixby. Solving real-world linear programs: a decade and more of progress. *Operations Research*, 2001.

Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, 1999.

A. Canutescu, A. Shelenkov, and R. L. Dunbrack. A graph-theory algorithm for rapid protein side-chain prediction. *Protein Sci*, 12(9):2001–2014, 2003.

R. Cowell. Advanced inference in Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1998.

R. L. Dunbrack and M. Kurplus. Back-bone dependent rotamer library for proteins: Application to side-chain predicrtion. *J. Mol. Biol*, 230:543–574, 1993.

J. Feldman, D. Karger, and M. J. Wainwright. LP decoding. In *Allerton Conference on Communication, Control, and Computing*, 2003.

P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. In *Proceedings of IEEE CVPR*, pages 261–268, 2004.

A. S. Fraenkel. Protein folding, spin glass and computational complexity. In *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers, held at the University of Pennsylvania, June 23 – 25, 1997*, pages 175–191, 1997.

C. L. Kingsford, B. Chazelle, and M. Singh. Solving and analyzing side-chain positioning problems using linear and integer programming. *Bioinformatics*, 21(7):1028–1039, 2005. doi: 10.1093/bioinformatics/bti144.

V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. In *Proceedings AI Stats*, 2005.

V. Kolmogorov and M. J. Wainwright. On the optimality of tree-reweighted max-product message passing. In *Uncertainty in Artificial Intelligence (UAI)*, 2005.

V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

T. Kortemme, A. V. Morozov, and D. Baker. An orientation-dependent hydrogen bonding potential improves prediction of specificity and structure for proteins and protein-protein complexes. *Journal of Molecular Biology*, 326(4):1239–1259, 2003.

B. Kuhlman and D. Baker. Native protein sequences are close to optimal for their structures. *PNAS*, 97(19):10383–10388, 2000.

T. Lazaridis and M. Karplus. Effective energy function for proteins in solution. *Proteins: Structure, Function, and Genetics*, 35(2):133–152, 1999.

R. Marinescu, K. Kask, and R. Dechter. Systematic vs. non-systematic algorithms for solving the MPE task. In *Proceedings of Uncertainty in Artificial Intelligence (UAI 2003)*, 2003.

T. Meltzer, C. Yanover, and Y. Weiss. Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation. In *Proceedings International Conference on Computer Vision (ICCV)*, 2005.

K. Murphy. The bayes net toolbox for matlab. *Computing Science and Statistics*, 33, 2001.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

N. A. Pierce and E. Winfree. Protein Design is NP-hard. *Protein Eng.*, 15(10):779–782, 2002.

C. A. Rohl, C. E. M. Strauss, D. Chivian, and D. Baker. Modeling structurally variable regions in homologous proteins with rosetta. *Proteins: Structure, Function, and Bioinformatics*, 55(3): 656–677, 2004.

E. G. Santos. On the generation of alternative explanations with implications for belief revision. In *UAI*, 1991.

D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Compter Vision*, 2002.

D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, volume 1, pages 195–202, 2003.

Y. Shimony. Finding the MAPs for belief networks is NP-hard. *Aritifical Intelligence*, 68(2):399–410, 1994.

A. G. Street and S. L. Mayo. Computational protein design. *Structure with folding and design*, 7: r105–r109, 1999.

J. Sun, H.-Y. Shum, and N.-N. Zheng. Stereo matching using belief propagation. In *ECCV*, volume 2, pages 510–524, 2002.

M. Tappen and W. T. Freeman. Graph cuts and belief propagation for stereo, using identical MRF parameters. In *ICCV*, 2003.

E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

M. J. Wainwright, T. Jaakkola, and A. S. Willsky. MAP estimation via agreement on (hyper)trees: Message-passing and linear programming approaches. In *Allerton Conference on Communication, Control, and Computing*, 2002.

C. Yanover and Y. Weiss. Approximate inference and protein folding. *Advances in Neural Information Processing Systems*, 2002.

J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *IJCAI (distinguished lecture track)*, 2001.