Distributed Anytime MAP Inference

Joop van de Ven & Fabio Ramos

T: +61 2 9114 0639, +61 2 9036 7058
E: j.vandeven@acfr.usyd.edu.au, f.ramos@acfr.usyd.edu.au

Technical Report ACFR-TR-2010-001

12 October 2010
Released 13 October 2010
Abstract

We present a distributed anytime algorithm for performing MAP inference in graphical models. We devise a new formulation for the problem as a linear program relaxation over edges. This results in a constraint structure that enables the use of the Dantzig-Wolfe decomposition principle. The subprograms are defined over individual edges and can be computed in a distributed manner, thus allowing the solution of very large networks whose state space does not even fit in memory. The decomposition master program is guaranteed to compute the optimal solution in a finite number of iterations where the solution is monotonically converging. Experimental results show that our algorithm is an order of magnitude faster than current state of the art methods while achieving comparable accuracy.
## Contents

1 Introduction ........................................ 4

2 LP formulation ..................................... 5
   2.1 Solution consistency ........................... 5

3 Decomposition ..................................... 6
   3.1 Initialisation ................................ 7
   3.2 Subprogram ................................... 7
   3.3 Master program ............................... 7
   3.4 Optimal solution ............................. 8

4 Experiments ...................................... 8

5 Discussion and future work ...................... 9
**1 Introduction**

Undirected graphical models are powerful tools for modelling many real world problems. They have been successfully applied to a diversity of problems such as: image processing [1], protein design [2], and text labelling [3]. One desirable operation in these models is to infer the most probable configuration, also known as the maximum a posteriori (MAP) problem. For tree-structured graphs, there are algorithms guaranteed to compute the globally optimal MAP solution in polynomial time (see for example: [4, 5]).

For arbitrary graphs however, finding the MAP configuration has been proven to be NP-hard [6]. For such graphs approximation algorithms are required to generate solutions in a computationally feasible time-span. One particularly popular algorithm is based on the max-product belief propagation due to Pearl [5]. In belief propagation each node sends messages to each of its neighbours. The messages convey what a node believes its neighbours state should be given its own state. Max-product is exact for tree-structured graphs; for arbitrary graphs the algorithm can be adapted to run for a number of iterations in a cyclic manner. The procedure is known as loopy belief propagation (LBP) [7]. LBP can be stopped when the change in messages falls below some threshold or a maximum number of iterations has been reached. Despite not having any convergence or optimality guarantees, the algorithm has been shown to generate good results for a large number of practical problems.

Several variants of max-product belief propagation have been proposed. Generalised belief propagation [8] extends the message passing from pairs of connected nodes to higher order cliques resulting in better approximations. Tree-reweighted max-product (TRMP, see [9]) on the other hand decomposes the original graph into a convex combination of tree-structured graphs. The tree-structured graphs guarantee efficient computations, while the convex combination allows the computation of an upper bound on the optimal solution. However, Yanover et al. [10] showed that TRMP fails to solve many difficult real world problems.

TRMP has strong connections to the linear programming message passing (LPMP) algorithm proposed by Globerson and Jaakkola [11]. LPMP is defined as a block coordinate descent in the dual of a linear program confined by the marginal polytope. The advantage of this approach is a strong convergence guaranteed. The disadvantage is that the marginal polytope has an exponential number of constraints preventing computationally efficient solutions unless the polytope is approximated (for example by the local marginal polytope). In particular, the method developed by Sontag et. al. [2, 12] is considered the state of the art in MAP inference. Their algorithm starts with the local marginal polytope and iteratively clusters are added until an integer solution is obtained.

Ravikumar and Lafferty [13] formulate the MAP problem as a quadratic programming (QP) relaxation. QP relaxations are a more natural fit to the MAP problem as pairwise potentials in a graph are quadratic in nature. For many practical problems however, the QP relaxation is non-convex thus requiring further approximations in order to produce an algorithm that is solvable in polynomial time. Despite this convex approximation, the QP relaxation has been shown to often generate better results compared to a LP formulation [14]. The drawback to using a QP relaxation is that it requires memory squared in the number of states, so for medium to large scale problems, this approach quickly becomes impractical.

In this paper we propose a novel linear programming relaxation to the MAP inference problem in a graphical model, $G = (V, E)$. Starting from a quadratic formulation over the nodes $s \in V$ (analogous to [13]) the problem is transformed into an integer formulation over the edges $(s, t) \in E$, which is subsequently relaxed into a LP. The transformation from quadratic to linear increases both the number of variables ($O(|V|^k) \rightarrow O(|E|k^2)$) as well as constraints ($O(|V|) \rightarrow O(|E| + \sum_{s \in V} (|\mathcal{N}(s)| - 1)k)$). For ease of notation, each node of the graph has $k$ states, while $\mathcal{N}(s)$ are the neighbours of node $s$. For problems where the proposed LP relaxation fits into memory, any LP solver may be used to compute the optimal solution. For medium to large scale problems there may not be sufficient memory available to solve the LP in one step. However, as will be shown, there is a structure to the constraints that allows application of the Dantzig-Wolfe decomposition principle [15]. The decomposition principle reformulates the linear program into a number of subprograms, one for each edge in our case, together with a master program. The subprograms can be solved independently and distributed while the master program solves for the optimal solution using a reduced number of states. The algorithm is solved iteratively, solutions to the subprograms are used to update the master program and vice versa. At each iteration the decomposition principle guarantees that the solution will be closer to the optimal solution, i.e. an anytime algorithm. We show that this approach achieves significant speed improvements over current state of the art methods with the advantage of being intrinsically distributed.
2 LP formulation

An undirected graphical model $G = (V, E)$ represents a probability distribution $p_G(x_1, \ldots, x_N)$ over $N = |V|$ variables. The vertices $s \in V$ of the graph index the random variables $x_s$ of the distributions, while the edges $(s, t) \in E$ of the graph capture relationships between variables $x_s$ and $x_t$. Let $C$ be the set of all cliques of the graph. The distributions must factor as a product of clique potentials $\phi_c(X_c)$, where $c \in C$ and $X_c$ are the variables of the clique. Yedidia et al. [8] showed that, without loss of generality, it is possible to assume that the graph is a pairwise Markov random field, i.e. the set of cliques $C_M = \{(s, t) \in E\}$. As a result, the log of the distribution of $X = \{x_1, \ldots, x_N\}$, for the graph $G$ with potentials $\Phi = \{\phi_c | c \in C_M\}$ is given by

$$
\log p_G(X; \Phi) = \sum_{s \in V} \phi_s(x_s) + \sum_{(s, t) \in E} \phi_{st}(x_s, x_t) - C,
$$

where $C$ is the log of the partition function. In the remainder we shall only consider variables $x_s$ that take on values from a finite discrete set $\mathcal{X}_s$. Each $x_s$ is a vector of length $|\mathcal{X}_s|$ with elements $x_s^k \in \{0, 1\}$ and $\sum_s x_s^k = 1$, i.e. a discrete graphical model. Furthermore, without loss of generality, we shall include the local potentials in the pairwise potentials. For discrete graphical models, the combined pairwise potential may then be expressed as:

$$
Q_{st} = \phi_{st} + (e\phi_s^T|N(s)| + (\phi_t e^T)/|N(t)|,
$$

where $N(s)$ is the set of neighbours of node $s$. Division by the number of neighbours distributes the local potential evenly over the pairwise potentials while leaving the MAP value unaltered. The vector $e$ is an appropriately sized vector of 1s. This leads to the following quadratic integer MAP problem,

$$
X_{MAP} = \arg\max_X \log p_G(X; \Phi) = \arg\max_X \sum_{(s, t) \in E} x_s^T Q_{st} x_t.
$$

For small scale problems equation 3 may be solved by a quadratic programming relaxation. However, for medium or large scale problems the resulting relaxation will quickly become too large to fit into memory. In addition, there is no convexity guarantee for $Q_{st}$; preventing computationally efficient algorithms from being used without further convex approximations. Instead we shall reformulate the quadratic objective function into a LP by substitution. The substitution transforms the problem from an optimisation over the nodes into one over the edges with a constraint structure that allows the LP to be solved distributively.

For each edge $(s, t) \in E$, define the edge variable

$$
y_{st} = (x_s^k x_t^k | i = 1, \ldots, |\mathcal{X}_s|, j = 1, \ldots, |\mathcal{X}_t|, (s, t) \in E)
$$

which, by the discrete nature of $x_s$ and $x_t$, has elements $y_{st}^k \in \{0, 1\}$ and $\sum_k y_{st}^k = 1$ ($k = 1, \ldots, |\mathcal{X}_s|, |\mathcal{X}_t|$). Equally, the cost $c_{st}$ can be constructed from $Q_{st}$ by ordering the elements of $Q_{st}$ into a vector corresponding to the elements of $y_{st}$. The LP relaxation to the MAP problem may then be formulated as

$$
\begin{align*}
\text{Maximise} & \quad \sum_{(s, t) \in E} c_{st}^T y_{st} \\
\text{Subject to} & \quad A_{st} y_{st} - A_{st} y_{su} = 0 \quad \forall s \in V, t \in N(s), \forall u \in N(s), u \neq t \\
& \quad \sum y_{st} = 1 \quad \forall (s, t) \in E \\
& \quad 0 \leq y_{st} \leq 1 \quad \forall (s, t) \in E.
\end{align*}
$$

The constraints defined by the matrix coefficients $A_{st}$ are the consistency constraints. The elements of $A_{st}$ are 0 or 1, such that $A_{st}^i y_{st} = x_s^i$ where $A_{st}^i$ is the $i$-th row of $A_{st}$. Consistency constraints are discussed in more detail in section 2.1. The second set of constraints express a uniqueness of solution for each edge; only one element of the edge variable $y_{st}$ may be active. While the third set of constraints capture the relaxation from an integer program to a linear program.

2.1 Solution consistency

The LP of equation 5 is defined over the edges, as such constraints are required to ensure the solution of the LP remains consistent in the node variables.

**Definition** Let $m_{st} = A_{st} y_{st}$ be a marginal, for node variable $x_s$, of the edge variable $y_{st}$. 
Proposition 2.1 The solution for the edge variables \( \{y_{st} | \forall t \in \mathcal{N}(s)\} \) is consistent in the node variable \( x_s \) when the marginals \( \{m_{st} | \forall t \in \mathcal{N}(s)\} \) are all equal.

Proof The proof can be obtained by simple substitution of \( y_{st}^h = x_s^i x_t^j \) and \( \sum x_s^i = 1 \).

Consistency constraints are specified over pairs of edges, i.e. the difference between pairs of marginals \( m_{si} \) and \( m_{sj} \). For a given node \( s \) one edge is used as the reference edge; edge \((s, t)\) in equation 5. All consistency constraints are specified relative to the reference edge resulting in a minimum of constraints generated. Subsequently solving equation 5 will result in a solution for the edge variables. The mapping from \( y_{st} \) to the node variables \( x_s \) is given by the following proposition.

Proposition 2.2 If the linear program of equation 5 has a feasible solution, then the mapping from \( y_{st} \) to \( x_s \) is given by \( x_s = m_{st} \) for any \( t \in \mathcal{N}(s) \).

Proof The equality \( x_s = m_{st} \) follows directly from the definitions of \( y_{st} \) and \( m_{st} \). Proposition 2.1 permits any \( t \in \mathcal{N}(s) \) provided the solution is consistent. Solution consistency, and therefore proposition 2.1, is ensured by virtue of a feasible solution; all constraints are met.

3 Decomposition

The Dantzig-Wolfe decomposition principle [15] allows linear programs, with a special block-matrix structure, to be broken up into a number of independent linear subprograms. The subprograms are iteratively adjusted to take account of interconnections due to a master program. The reader is referred to chapter 10 of [15] for a detailed discussion and proofs. In this section we shall confine ourselves to providing an interpretation of the principle in the context of MAP inference in graphical models.

The block-angular system,

\[
\begin{align*}
\text{Maximise} & \quad c^T_1 y_1 + \ldots + c^T_K y_K \\
\text{Subject to} & \quad B_1 y_1 + \ldots + B_K y_K = b \\
& \quad f_1 y_1 = f_1 \\
& \quad \vdots \\
& \quad f_K y_K = f_K \\
& \quad y_i \geq 0 \\
& \quad i = 1, \ldots, K,
\end{align*}
\]

allows decomposition to be applied to the MAP problem for \( K = |E| \). The matrices \( \{B_i | i = 1, \ldots, K\} \) form the coupling constraints; they capture interconnections between the subprograms. While the constraints unique to each subprogram are constructed from \( \{F_i | i = 1, \ldots, K\} \). The constraints of the MAP problem, equation 5, have the block-angular structure. \( B_1 \) is constructed from the set \( \{A_{st}\} \) for edge \((s, t)\); appropriately padded with 0s to ensure all \( B \) have the same number of rows. The subprogram constraints are the uniqueness of solution constraints; \( F_{st} = \sum y_{st} \).

The decomposition principle exploits the resolution theorem [15]. Briefly, the resolution theorem states that every feasible solution of the convex polyhedral set \( Ax = b, x \geq 0 \) can be represented as a convex combination of its extreme points\(^1\). see Theorem 10.5 of [15] for more details. Using the resolution theorem, equations of the form of equation 6 can be transformed into a master program that maximises a convex combination of extreme points together with \( K \) subprograms that generate extreme points at each iteration.

We shall now present the steps of the algorithm, each of the steps are discussed in more detail in the sections to follow:

1. Initialise the algorithm (section 3.1) to find an initial basic feasible solution.
2. Solve the master program using all columns corresponding to the initial basic feasible solution.
   This provides the interconnections in the form of the simplex multipliers \( (\pi, \gamma) \); see section 3.3 for more details.
3. Solve all subprograms using the current simplex multipliers (section 3.2).
4. Add columns to the master program according to optimality of subprogram solutions and corresponding column cost. Solve the master program to obtain new interconnections \( (\pi, \gamma) \); section 3.3.

\(^1\)Normalised extreme homogeneous solutions are omitted as our subprograms cannot generate these.
5. If the master program has found the optimal solution go to step 6, if not go to step 3.

6. Transform the master program’s solution to the solution of equation 5 and perform rounding if required, see section 3.4 for more details.

### 3.1 Initialisation

The aim of initialisation is to find an initial basic feasible solution. One common approach to initialising a Dantzig-Wolfe decomposition is to find the maxima of each subprogram individually, using the actual costs. The resulting solutions are used to start the master program. However, in our case this generally means that the consistency constraints are violated, thus preventing the decomposition from even starting. Instead the procedure of algorithm 1 is used to find a initial basic feasible solution in one step.

#### Algorithm 1 Pseudo-code of algorithm initialisation.

Input: Graph $G = (V, E)$, local potentials $\phi_s, \forall s \in V$ and pairwise potentials $\phi_{st}, \forall (s, t) \in E$

Output: Initial basic feasible solutions \{\(\tilde{y}_{st}\) | \((s, t) \in E\)} for \(s \in V\)

def initialisation(G, \phi, \phist):
    for \(s \in V\) do
        \(\tilde{\phi} \leftarrow \phi_s\)
        for \(t \in \mathcal{N}(s)\) do
            \(\tilde{\phi} \leftarrow \tilde{\phi} + \sum_{x_t} \phi_{st}\)
        end for
        \(\tilde{x_s} \leftarrow \arg\max_{x_s}(\tilde{\phi})\)
    end for
    for \((s, t) \in E\) do
        \(\tilde{y}_{st} \leftarrow (\tilde{x_i}_s \tilde{x_j}_t | i = 1, \ldots, |X_s|, j = 1, \ldots, |X_t|)\)
    end for

As can be seen, a node’s initial solution $\tilde{x}_s$ is found by maximising over the sum of local and marginalised pairwise potentials. Once the initial solutions for each node have been found, they are mapped to their equivalent subprogram initial basic feasible solutions, $\tilde{y}_{st}$. Since the initial solutions $\tilde{y}_{st}$ are based on node solutions, the consistency constraints are always met. The subprograms’ initial basic feasible solutions are subsequently used to get the decomposition master program started.

Selection of the initial solution $\tilde{x}_s$ based on the maximum from both node and connected edges provides an initial basic feasible solutions $\tilde{y}_{st}$ based on the maxima of the local neighbourhood. In other words, the algorithm starts with a local (node) view of the problem. With each iteration that view will be aligned more closely to the global (and optimal) view.

### 3.2 Subprogram

For inference in a graph, the subprograms maximise a linear program over the edges as follows:

\[
\begin{align*}
\text{Maximise} & \quad c_{st}^T y_{st} - B_{st}^T \pi \\
\text{Subject to} & \quad \sum y_{st} = 1 \\
& \quad 0 \leq y_{st} \leq 1.
\end{align*}
\]

(7)

As can be seen from equation 7, the cost is the actual cost of the edge, $c_{st}$, adjusted by the current state of the interactions, $B_{st}^T \pi$. Where $B_{st}^T$ are the concatenated consistency constraints and $\pi$ are the corresponding simplex multipliers. The adjusted cost finds the edge’s maximum based on the current global state of the algorithm. It is however, not necessary to invoke a LP solver for each subprogram. There are two constraints for each edge; they express the uniqueness of solution ($y_{st}^k \in \{0, 1\}$ and $\sum_k y_{st}^k = 1$). A solution to the subprograms can therefore be found by a simple maximisation over a vector; i.e. the solution is always an extreme point.

Let the solution $y_{st,i}$ represents the optimal solution for edge $(s, t)$ at iteration $i$. If $c_{st}^T y_{st,i} - B_{st}^T \pi \neq \gamma$ then this solution is globally sub-optimal and it may be incorporated into the master program, provided it has not previously been incorporated. In case of a tie (multiple solutions with the same maximum) we select the solution with the lowest index $k$; analogous to Bland’s rule [16].

### 3.3 Master program

The purpose of the master program is twofold. First, it generates the interconnections in the form of the simplex multipliers $\pi$ and $\gamma$. Second, any feasible solution to the master program can be transformed
into a solution of the original LP, equation 5. At each iteration of the algorithm columns are added to the master program - depending on the optimality of the solutions of the subprograms. The added columns allow the master program to update the interconnections based on the subprogram solutions.

For the MAP inference problem, the master program is defined as shown in equation 8,

$$\begin{align*}
\text{Maximise} & \quad \sum_{(s,t) \in E} \sum_{i \in L_{st}} g_{st}^i \alpha_{st}^i \\
\text{Subject to} & \quad \sum_{(s,t) \in E} \sum_{i \in L_{st}} \alpha_{st}^i \hat{y}_{st,i} = 0 \\
& \quad \sum_{i \in L_{st}} \alpha_{st}^i = 1 \quad \forall (s,t) \in E \\
& \quad \alpha_{st} \geq 0 \quad \forall (s,t) \in E.
\end{align*}$$

Where $L_{st}$ is the set of iteration indices, of edge $(s,t)$, for which columns have been added to the master program. The element $g_{st}^i = c_{st}^i \gamma_{st,i}$ is the cost of the subprogram solution at iteration $i$. While $\alpha_{st}^i = B_{st} \hat{y}_{st,i}$ is the column of corresponding consistency constraints. The elements $\alpha_{st}^i$ are the convexity variables from the resolution theorem. They have a particularly elegant interpretation for the MAP problem; they represent the likelihood of the subprogram solutions $\hat{y}_{st,i}$. Note that $\pi$ consists of the simplex multipliers corresponding to the consistency constraints while $\gamma$ are the simplex multipliers of the convexity variables.

Up to $|E|$ columns may be added to the master program - one for each subprogram. However, not all of these prospective columns will aid in finding a solution. Quite to the contrary, often they will add unnecessary complexity to the master program. Instead of adding all prospective columns to the master program, a limited number of columns are added at each iteration. Columns are selected based on their cost; at iteration $i$ only columns corresponding to maximal costs $g_{st}^i$ are added (similar to pivoting in the simplex method). The experiments limit the number of columns to add to a maximum of 200.

### 3.4 Optimal solution

The Dantzig-Wolfe decomposition has the desirable property that it is guaranteed to converge in a finite number of iterations (Theorem 10.4 from [15]). Once converged, the solution to equation 5 can be found from the convex variables $\alpha_{st}^i$ and the corresponding subprograms’ optimal solutions $\hat{y}_{st,i}$,

$$\hat{y}_{st} = \sum_{i \in L_{st}} \alpha_{st}^i \hat{y}_{st,i}. \quad \quad \quad (9)$$

The globally optimal solution $\bar{y}_{st}$ is the sum of the subprograms’ optimal solutions, scaled by their corresponding convex variable (or likelihood). The $\hat{y}_{st}$ can be mapped back to a corresponding optimal node solution $\bar{x}$ (see section 2.1). Rounding may have to be applied to $\bar{x}$ to find an integer solution. The rounding scheme used in the experiments is deterministic. As each $\bar{y}_{st,i}$ is scaled by its likelihood, we therefore simply set the maximal element $\hat{y}_{st}^i$ to 1 and all others to zero.

### 4 Experiments

The performance of the proposed algorithm is measured on the Rosetta Side-Chain Prediction dataset [10]. Following Sontag et al. [2], we apply our algorithm to the 30 graphs that most inference algorithms, such as TRMP, are unable to solve. Note that, to date, the method by Sontag et al. is the only method capable of solving all these graphs and is thus our main benchmark for comparison. We are able to solve all 30 graphs to within 3.3% (on average) of their true MAP value, see also table 2. The method by Sontag et al. solves all graphs to their true value, however, this comes at a significant computational expense. In their work [2] they state that the running times vary between 1 minute and 1 hour, with over half the graphs solved in under 9 minutes. The proposed algorithm is significantly faster; half the graphs are solved under 53 seconds, with 95% of the graphs solved in less than 8 minutes. The largest graph was solved in just over 23 minutes while the fastest solution was obtained in less than 8 seconds (the average running time per graph is under 3 minutes). Various timing statistics are given in table 1; all results are obtained on a 3Ghz CPU, the decomposition code is implemented in Matlab while the LP solver is CPLEX.

A second important observation, concerning the timing results of table 1, can be made. The statistics for the dataset are obtained in a serial manner. No parallelism is applied to the subprograms as each graph fits into memory. There is therefore no practical need to distribute the implementation; graphs that do not fit into memory can be distributed. The subprograms, however, take on average 37% of the total execution time (see also figure 1(d) for timing results of a single graph). This means that significant timing gains can be made from parallelising the implementation even for graphs that do fit into memory.
<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (s)</td>
<td>7.735</td>
<td>1404.005</td>
<td>176.645</td>
<td>299.579</td>
</tr>
<tr>
<td>Master Program (%)</td>
<td>42.410</td>
<td>71.594</td>
<td>58.118</td>
<td>8.335</td>
</tr>
<tr>
<td>Sub Program (%)</td>
<td>23.734</td>
<td>51.906</td>
<td>37.293</td>
<td>7.574</td>
</tr>
<tr>
<td>Rounding &amp; Overhead (%)</td>
<td>1.177</td>
<td>7.746</td>
<td>4.589</td>
<td>1.802</td>
</tr>
</tbody>
</table>

Table 1: Timing Statistics.

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP Value Approximation (%)</td>
<td>0.026</td>
<td>38.210</td>
<td>3.313</td>
<td>7.879</td>
</tr>
<tr>
<td>MAP Configuration Approximation (%)</td>
<td>2.033</td>
<td>8.276</td>
<td>4.984</td>
<td>1.825</td>
</tr>
<tr>
<td>Fractional Solution (%)</td>
<td>0.434</td>
<td>10.435</td>
<td>3.088</td>
<td>2.094</td>
</tr>
<tr>
<td>Fractional Approximations (%)</td>
<td>0</td>
<td>66.667</td>
<td>33.581</td>
<td>16.171</td>
</tr>
</tbody>
</table>

Table 2: Solution Quality Statistics.

From table 2 it can be seen that, on average, the MAP value is within 3.3% of its true value. This can also be observed, for a single graph, from figure 1(a); the figure also shows the desirable monotonic convergence property of the algorithm. An analogous observation can be made for the MAP configuration found by the algorithm. From table 2 we can see that the MAP configuration is 95% correct on average. Figure 1(b) shows that the MAP configuration, generally speaking, improves with each iteration. A final observation with respect to the solution quality is that just over 3% of the MAP configuration has a non-integer solution, for which rounding is subsequently required. However, these nodes account for about a third of the MAP configuration approximations. This would seem to suggest that the relaxation from integer to linear program is the major source for the approximations.

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP Variables</td>
<td>92151</td>
<td>1569834</td>
<td>460245</td>
<td>379522</td>
</tr>
<tr>
<td>Master Program Variables</td>
<td>6322</td>
<td>183082</td>
<td>41608</td>
<td>43069</td>
</tr>
<tr>
<td>Percentage of Variables (%)</td>
<td>3.987</td>
<td>13.081</td>
<td>8.164</td>
<td>2.447</td>
</tr>
</tbody>
</table>

Table 3: Variable Usage Statistics.

Finally, from table 3 it can be seen that, on average, just over 8% of the variables of the original LP (equation 5) are required to solve the master program. This can also be observed from figure 1(c); initially the 200 most promising variables are added per iteration. Once the algorithm gets close to the optimal solution fewer variables are needed. Couple this significant reduction in the number of variables with the fact that the constraints for the master program are very sparse, and it becomes clear that the proposed algorithm can be used to effectively deal with large scale problems - very little memory is required to define and solve the master program.

5 Discussion and future work

This paper presented a linear programming formulation of the MAP inference problem in undirected graphical models. Unlike other linear (or quadratic) programming formulations, ours is defined over edge variables instead of over node variables. The advantage of such a formulation is that the LP constraints allow a decomposition into a number of subprograms (one for each edge) together with a small master program. The subprograms can be distributed over a network to allow large-scale problems to be solved efficiently. In addition, the master program iteratively gets closer to the optimal solution resulting in an anytime algorithm for performing MAP inference. Experimental results show that the algorithm is computationally faster than current state of the art algorithms while finding comparable solutions.

Moving forward there are a number of areas where further research will help improve both the computational efficiency as well as the quality of the solution. At present the consistency constraints are
defined over pairs of edges for a given node. Computationally much can be gained if the consistency constraints can be defined for all edges around a node simultaneously, instead of over pairs.

A second area of research is in guiding the algorithm, both in the subprograms as well as the master program. The proposed guidance is based on Bland’s rule and column cost; analogous to the simplex method. Guidance based on constraint satisfaction or the Largest-Coefficient rule [17] may be considered in the future, as well as determining the number of variables to add based on the state of the algorithm.

Figure 1: Analysis of the algorithm for the 1ug6 graph. (a) Negative MAP value (due to CPLEX minimisation). (b) Number of nodes with an approximate solution (after rounding). (c) Number of master program columns (variables). (d) Execution time.

References


