Exploiting Within-Clique Factorizations in Junction-Tree Algorithms

Julian J. McAuley
NICTA - Australian National University

Tibério S. Caetano
NICTA - Australian National University

Abstract

We show that the expected computational complexity of the Junction-Tree Algorithm for *maximum a posteriori* inference in graphical models *can be improved*. Our results apply whenever the potentials over maximal cliques of the triangulated graph are factored over subcliques. This is common in many real applications, as we illustrate with several examples. The new algorithms are easily implemented, and experiments show substantial speed-ups over the classical Junction-Tree Algorithm. This enlarges the class of models for which exact inference is efficient.

1 INTRODUCTION

It is well-known that exact inference in *tree-structured* graphical models can be accomplished efficiently by message-passing operations following a simple protocol making use of the distributive law [Aji and McEliece, 2000]. It is also well-known that exact inference in *arbitrary* graphical models can be solved by the Junction-Tree Algorithm; its efficiency is determined by the size of the maximal cliques after triangulation, a quantity related to the treewidth of the graph.

Figure 1 illustrates an attempt to apply the Junction-Tree Algorithm to some graphical models containing cycles. If the graphs are not chordal ((a) and (b)), they need to be triangulated, or made chordal (red edges in (c) and (d)). Their clique-graphs are then guaranteed to be *Junction-Trees*, and the distributive law can be applied with the same protocol used for trees (see [Aji and McEliece, 2000] for an excellent tutorial on exact inference in arbitrary graphs). Although the models in this example contain only pairwise factors, triangulation has increased the size of their maximal cliques, making exact inference substantially more expensive. Hence approximate solutions in the original graph (such as Loopy Belief-Propagation, or inference in a Loopy Factor-Graph) are often preferred over an exact solution via the Junction-Tree Algorithm.

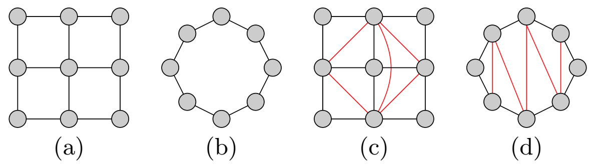


Figure 1: The models at left ((a) and (b)) can be triangulated ((c) and (d)) so that the Junction-Tree Algorithm can be applied. Despite the fact that the new models have larger maximal cliques, the corresponding potentials are still factored over pairs of nodes.

In this paper, we exploit the fact that the maximal cliques (after triangulation) often have potentials that factor over subcliques, as illustrated in Figure 1. We will show that whenever this is the case, the expected computational complexity of exact inference *can be improved* (both the asymptotic upper bound and the actual runtime). This will increase the class of problems for which exact inference is tractable.

This is not to be confused with optimizations produced by *Factor Graphs* [Kschischang et al., 2001]. If applied to the above examples, the resulting Factor Graphs would contain cycles and would therefore produce inexact solutions in general. Instead, we work at the level of *Junction-Trees* arising from triangulated graphs, enabling us to leverage within-clique factorizations while performing *exact* inference.

A core operation encountered in the Junction-Tree Algorithm is that of finding the index that chooses the largest product amongst two lists of length N:

$$\hat{i} = \underset{i \in \{1 \ldots N\}}{\operatorname{argmax}} \{\mathbf{v}_a[i] \times \mathbf{v}_b[i]\}. \tag{1}$$

Our results stem from the realisation that while (eq. 1) appears to be a *linear* time operation, it can be decreased to $O(\sqrt{N})$ (in the expected case) if we know the permutations that sort $\mathbf{v}_a$ and $\mathbf{v}_b$.

Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 6 of JMLR: W&CP 6. Copyright 2010 by the authors.

1.1 SUMMARY OF RESULTS

A selection of the results to be presented in the remainder of this paper can be summarized as follows.

We are able to lower the asymptotic expected running time of the Junction-Tree Algorithm for *any* graphical model whose clique-potentials factorise into lower-order terms; we always obtain the same solution as the traditional Junction-Tree Algorithm, i.e., no approximations are used. For cliques composed of pairwise factors, we achieve an expected speed-up over the existing approach of *at least* $\Omega(\sqrt{N})$ (assuming N states per node); for cliques composed of K-ary factors, the expected speed-up becomes $\Omega(\frac{1}{K}N^{\frac{1}{K}})$ (Ω denotes an *asymptotic lower-bound*).

As an example, we can exactly compute the *maximum a posteriori* (MAP) states of a ring-structured model (see Fig. 1(b)) with M nodes in $O(MN^2\sqrt{N})$; in contrast, Loopy Belief-Propagation takes $\Theta(MN^2)$ *per iteration*, and the exact Junction-Tree Algorithm takes $\Theta(MN^3)$ by triangulating the graph (Θ denotes an *asymptotically tight bound*).

The expected-case improvement is achieved when the conditional densities of different factors (with respect to their shared variables) have independent order-statistics; if their order-statistics are positively correlated, we can obtain better performance than the expected case; if they are negatively correlated, we may obtain worse performance, though our algorithm is never asymptotically more expensive than the traditional Junction-Tree Algorithm.

Our results do not apply for every semiring $S(+,\cdot)$, but only to those whose 'addition' operation defines an order; we also assume that under this ordering, our 'multiplication' operator satisfies

$$a < b \wedge c < d \Rightarrow a \cdot c < b \cdot d. \tag{2}$$

Thus our results certainly apply for the *max-product* and *min-product* semirings (as well as *max-sum* and *min-sum*), but not for *sum-product*. Consequently, our approach is useful for computing MAP-states, but cannot be used to compute marginal distributions. We also assume that the domain of each node is *discrete*.

2 BACKGROUND

In belief-propagation algorithms, the message from a clique X to an intersecting clique Y is defined by

$$m_{X\to Y}(\mathbf{x}_{X\cap Y}) = \max_{\mathbf{x}_{X\setminus Y}}\{\Phi_X(\mathbf{x}_X)\prod_{Z\in\Gamma(X)\setminus Y} m_{Z\to X}(\mathbf{x}_{X\cap Z})\} \tag{3}$$

(where $\Gamma(X)$ returns the neighbours of the clique X). If such messages are computed after Y has received messages from all of its neighbours except X (i.e., $\Gamma(X)\setminus Y$), then this defines precisely the update scheme used by the Junction-Tree Algorithm. The same update scheme is used for Loopy Belief-Propagation, though it is done iteratively in a randomized fashion. MAP-states are computed in a similar fashion, except that the messages from *all* neighbours are included in (eq. 3).

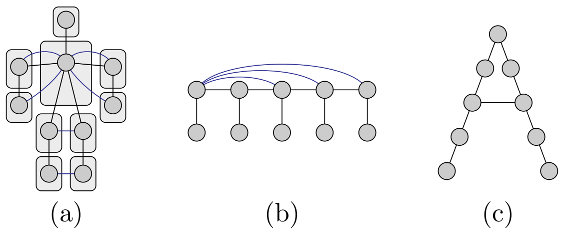


Figure 2: (a) A model for pose reconstruction from [Sigal and Black, 2006]; (b) A 'skip-chain CRF' from [Galley, 2006]; (c) A model for deformable matching from [Coughlan and Ferreira, 2002]. Although the (triangulated) models have cliques of size three, their potentials factorize into pairwise terms.

Often, the clique-potential $\Phi_X(\mathbf{x}_X)$ will be decomposable into several smaller factors, i.e.,

$$\Phi_X(\mathbf{x}_X) = \prod_{F\subset X}\Phi_F(\mathbf{x}_F). \tag{4}$$

Some simple motivating examples are shown in Figure 2: a model for pose estimation from [Sigal and Black, 2006], a 'skip-chain CRF' from [Galley, 2006], and a model for shape matching from [Coughlan and Ferreira, 2002]. In each case, the triangulated model has third-order cliques, but the potentials are only pairwise. Other examples have already been shown in Figure 1; analogous cases are ubiquitous in many real applications (to be shown in Section 4, Table 1).

The optimizations we suggest shall apply to general problems of the form

$$m_M(\mathbf{x}_M) = \max_{\mathbf{x}_{X\setminus M}}\prod_{F\subset X}\Phi_F(\mathbf{x}_F), \tag{5}$$

of which (eq. 3) is a special case (where the messages are considered to be factors). Computing the solution in the naïve way (i.e., evaluating $\prod_{F\subset X}\Phi_F(\mathbf{x}_F)$ for every value of $\mathbf{x}_X$) takes $\Theta(N^{|X|})$, where N is the number of states per node, and $|X|$ is the size of the clique X (we assume that for a given $\mathbf{x}_X$, computing $\prod_{F\subset X}\Phi_F(\mathbf{x}_F)$ takes constant time, as our optimisations shall not modify this cost). There is some loosely related work that applies to the *sum-product* version of this problem, based on arithmetic circuits [Park and Darwiche, 2003], an idea closely related to Strassen's sub-cubic method for matrix-multiplication.

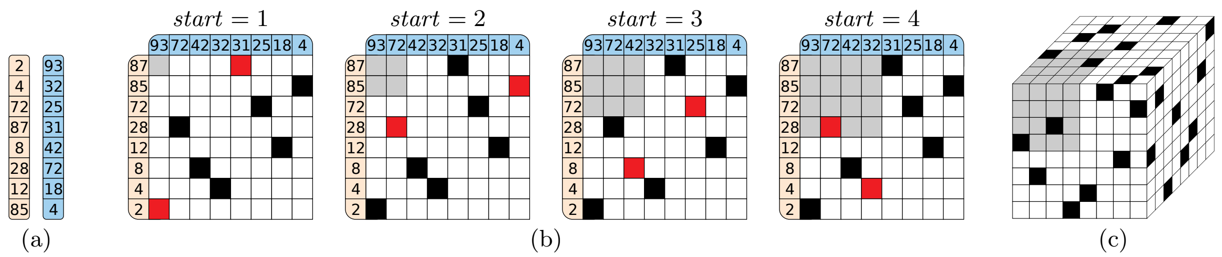


Figure 3: (a) The lists $\mathbf{v}_a$ and $\mathbf{v}_b$ before sorting. (b) Black squares show corresponding elements in the sorted lists ($\mathbf{v}_a[p_a[i]]$ and $\mathbf{v}_b[p_b[i]]$); red squares indicate the elements currently being read ($\mathbf{v}_a[p_a[\mathit{start}]]$ and $\mathbf{v}_b[p_b[\mathit{start}]]$). We can imagine expanding a gray box of size *start* $\times$ *start* until it contains an entry; note that the maximum is found during the first step. (c) In the version for three lists, we expand a gray box within a *cube*.

3 OUR APPROACH

To specify an efficient solution to (eq. 3), we first consider the simplest factorization: a clique of size three containing pairwise factors. Here we must compute

$$m_{i,j}(x_i, x_j) = \max_{x_k} \Phi_{i,j}(x_i, x_j)\Phi_{i,k}(x_i, x_k)\Phi_{j,k}(x_j, x_k). \tag{6}$$

For a particular value of $(x_i, x_j) = (a, b)$, we must solve

$$m_{i,j}(a, b) = \Phi_{i,j}(a, b) \times \max_{x_k} \underbrace{\Phi_{i,k}(a, x_k)}_{\mathbf{v}_a} \times \underbrace{\Phi_{j,k}(b, x_k)}_{\mathbf{v}_b}, \tag{7}$$

which we note is in precisely the form shown in (eq. 1).

This is sometimes referred to as 'funny' matrix multiplication, as (eq. 7) is equivalent to regular matrix multiplication with summation replaced by maximization. It is known to have a sub-cubic worst-case solution [Alon et al., 1997]; our approach does not improve the worst-case complexity, but gives far better *expected-case* performance than existing solutions.

As we have previously suggested, it will be possible to solve (eq. 7) efficiently if $\mathbf{v}_a$ and $\mathbf{v}_b$ are already sorted. We note that $\mathbf{v}_a$ will be reused for every value of x_j, and likewise $\mathbf{v}_b$ will be reused for every value of x_i. Sorting every row of $\Phi_{i,k}$ and $\Phi_{j,k}$ can be done in $\Theta(N^2 \log N)$ (for $2N$ rows of length N).

The following elementary lemma is the key observation required in order to solve (eq. 7) efficiently:

Lemma 1. *If the p^{th} largest element of $\mathbf{v}_a$ has the same index as the q^{th} largest element of $\mathbf{v}_b$, then we only need to search through the p largest values of $\mathbf{v}_a$,* ***and*** *the q largest values of $\mathbf{v}_b$; any values smaller than these cannot possibly contain the largest solution.*

This observation is used to construct Algorithm 1. Here we iterate through the indices starting from the largest values of $\mathbf{v}_a$ and $\mathbf{v}_b$, and stopping once both indices are 'behind' the maximum value found so far (which we then know is the maximum). This algorithm is demonstrated pictorially in Figure 3.

Algorithm 1 Find i that maximizes $\mathbf{v}_a[i] \times \mathbf{v}_b[i]$

```
Input: two vectors $\mathbf{v}_a$ and $\mathbf{v}_b$, and permutation functions $p_a$ and $p_b$ that sort them in decreasing order
    (so that $\mathbf{v}_a[p_a[1]]$ is the largest element in $\mathbf{v}_a$)
 1: Initialize: $start = 1$, $end_a = p_a^{-1}[p_b[1]]$, $end_b = p_b^{-1}[p_a[1]]$ {if $end_b = k$, the largest entry in $\mathbf{v}_a$ has
    the same index as the $k^{\text{th}}$ largest entry in $\mathbf{v}_b$}
 2: $best = p_a[1]$, $max = \mathbf{v}_a[best] \times \mathbf{v}_b[best]$
 3: if $\mathbf{v}_a[p_b[1]] \times \mathbf{v}_b[p_b[1]] > max$ then
 4:    $best = p_b[1]$, $max = \mathbf{v}_a[best] \times \mathbf{v}_b[best]$
 5: end if
 6: while $start < end_a$ {we do not check the stopping criterion for $end_b$; this creates some redundancy,
    which a more complex implementation avoids} do
 7:    $start = start + 1$
 8:    if $\mathbf{v}_a[p_a[start]] \times \mathbf{v}_b[p_a[start]] > max$ then
 9:       $best = p_a[start]$
10:       $max = \mathbf{v}_a[best] \times \mathbf{v}_b[best]$
11:    end if
12:    if $p_b^{-1}[p_a[start]] < end_b$ then
13:       $end_b = p_b^{-1}[p_a[start]]$
14:    end if
15:    {repeat Lines 8–14, interchanging $a$ and $b$}
16: end while {this takes expected time $O(\sqrt{N})$}
17: Return: $best$
```

A prescription of how Algorithm 1 can be used to solve (eq. 7) is given in Algorithm 2. Determining precisely the running time of Algorithm 1 (and therefore Algorithm 2) is not trivial, and will be explored in Section 3.2. We note that if the expected-case running time of Algorithm 1 is $O(f(N))$, then the time taken to solve Algorithm 2 shall be $O(N^2(\log N + f(N)))$. We will discuss the running time in Section 3.2, though for the moment we simply state the following theorem:

Theorem 2. *The* ***expected*** *running time of Algorithm 1 is $O(\sqrt{N})$, yielding a speed-up of at least $\Omega(\sqrt{N})$ in cliques containing pairwise factors.*

We can extend Algorithms 1 and 2 to cases where there are several overlapping terms in the factors. For instance, Algorithm 2 can be adapted to solve

$$m_{i,j}(x_i, x_j) = \max_{x_k, x_m} \Phi_{i,j}(x_i, x_j) \times \Phi_{i,k,m}(x_i, x_k, x_m) \times \Phi_{j,k,m}(x_j, x_k, x_m), \quad (8)$$

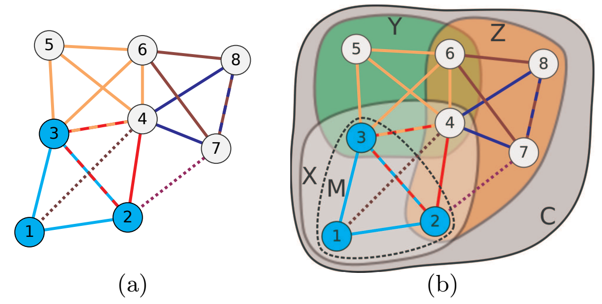
and similar variants containing three factors. Here both x_k and x_m are shared by $\Phi_{i,k,m}$ and $\Phi_{j,k,m}$. As the number of shared terms increases, so does the improvement to the running time. While (eq. 8) would take $\Theta(N^4)$ to solve using the naïve algorithm, it takes only $O(N^3)$ to solve using Algorithm 2. In general, if we have S shared terms, we create a new variable whose domain is their product space; the running time is then $O(N^2\sqrt{N^S})$, yielding a speed-up of $\Omega(\sqrt{N^S})$. over the naïve solution.

Algorithm 2 Compute the max-marginal of a 3-clique containing pairwise factors, using Algorithm 1

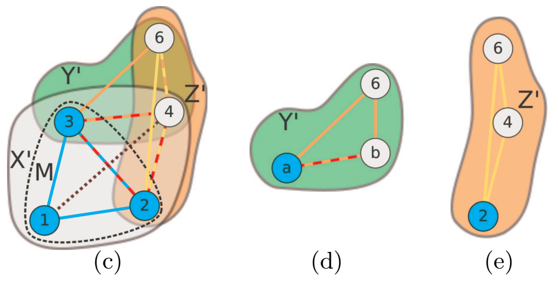
```
Input: a potential function $\Phi_{i,j,k}(x_i, x_j, x_k)$ with factors $\Phi_{i,j,k}(a,b,c) = \Phi_{i,j}(a,b) \times \Phi_{i,k}(a,c) \times \Phi_{j,k}(b,c)$
    whose max-marginal $m_{i,j}$ we wish to compute
 1: for $n \in \{1 \ldots N\}$ do
 2:     compute $\mathbf{P}^i[n]$ by sorting $\Phi_{i,k}(n, x_k)$
        {takes $\Theta(N \log N)$}
 3:     compute $\mathbf{P}^j[n]$ by sorting $\Phi_{j,k}(n, x_k)$
        {$\mathbf{P}^i$ and $\mathbf{P}^j$ are $N \times N$ arrays, each row of which is a permutation; $\Phi_{i,k}(n, x_k)$ and $\Phi_{j,k}(n, x_k)$ are
        functions over $x_k$, since $n$ is constant in this expression}
 4: end for {this loop takes $\Theta(N^2 \log N)$}
 5: for $(a, b) \in \{1 \ldots N\}^2$ do
 6:     $(\mathbf{v}_a, \mathbf{v}_b) = (\Phi_{i,k}(a, x_k), \Phi_{j,k}(b, x_k))$
 7:     $(p_a, p_b) = \big(\mathbf{P}^i[a], \mathbf{P}^j[b]\big)$
 8:     $best$ = $Algorithm1$ $(\mathbf{v}_a, \mathbf{v}_b, p_a, p_b)$ {$O(\sqrt{N})$}
 9:     $m_{i,j}(a, b) = \Phi_{i,j}(a, b)\Phi_{i,k}(a, best)\Phi_{j,k}(b, best)$
10: end for {this loop takes $O(N^2\sqrt{N})$}
11: Return: $m_{i,j}$
```

3.1 AN EXTENSION TO CLIQUES WITH ARBITRARY DECOMPOSITIONS

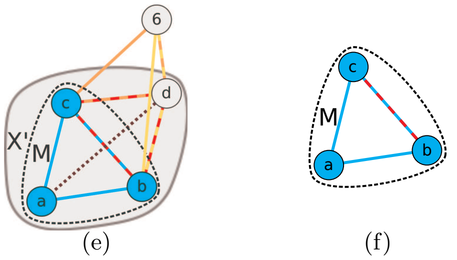
By similar reasoning, we can apply our algorithm in cases where there are more than three factors. We begin with the simplest case, in which our factors can be separated into three *groups* (which we note is always the case for pairwise factors). An illustrative example of such a clique is given in Figure 4(a), which we shall call G (assumed to be maximal in some triangulated graph). Each of the factors in this clique has been labeled using differently coloured edges, and the max-marginal we wish to compute has been labeled using coloured nodes. It is possible to split this graph into three groups X, Y, and Z, such that every factor is contained within a single group, along with the max-marginal we wish to compute.



(a) We begin with a set of factors (each coloured clique is a factor; dashed lines indicate overlapping factors, while dotted lines indicate pairwise factors), which are assumed to belong to some clique in our model; we wish to compute the max-marginal with respect to one of those factors (indicated using coloured nodes); (b) The factors are split into three groups, such that every factor is entirely contained within one of them (Algorithm 3, line 1).



(c) Any nodes contained in only one of the groups are marginalised (Algorithm 3, lines 2, 3, and 4); the problem is now very similar to that described in Algorithm 2, except that *nodes* have been replaced by *groups*; note that this essentially introduces maximal factors in Y' and Z'; (d) For every value $(a, b) \in \mathrm{dom}(x_3, x_4)$, $\Psi^Y(a, b, x_6)$ is sorted (Algorithm 3, lines 5–7); (e) For every value $(a, b) \in \mathrm{dom}(x_2, x_4)$, $\Psi^Z(a, b, x_6)$ is sorted (Algorithm 3, lines 8–10).



(e) For every $\mathbf{n} \in \mathrm{dom}(X')$, we choose the best value of x_6 by Algorithm 1 (Algorithm 3, lines 11–16); (f) The result is marginalised with respect to M (Algorithm 3, line 17).

Figure 4: Algorithm 3, explained pictorially. In this case, the most computationally intensive step is the marginalisation of Z (in step (c)), which takes $\Theta(N^5)$. However, the algorithm can actually be applied *recursively* to the group Z, resulting in an overall running time of $O(N^4\sqrt{N})$, for a max-marginal that would have taken $\Theta(N^8)$ to compute using the naïve solution.

Algorithm 3 Compute the max-marginal of G with respect to M, where G is split into three groups

```
Input: potentials $\Phi_G(\mathbf{x}) = \Phi_X(\mathbf{x}_X)\Phi_Y(\mathbf{x}_Y)\Phi_Z(\mathbf{x}_Z)$, where $M \subseteq X$ (see Fig. 4)
 1: Define: $X' = ((Y \cup Z) \cap X) \cup M$; $Y' = (X \cup Z) \cap Y$; $Z' = (Y \cup Z) \cap X$ {$X'$ contains the variables in $X$ that are shared by at least one other group; alternately, the variables in $X \setminus X'$ appear only in $X$ (sim. for $Y'$ and $Z'$)}
 2: compute $\Psi^X(\mathbf{x}_{X'}) = \max_{X \setminus X'} \Phi_X(\mathbf{x}_X)$
    {we are marginalising over those variables in $X$ that do not appear in any of the other groups (or in $M$); this takes $\Theta(N^{|X|})$ if done by brute force, but may also be done recursively}
 3: compute $\Psi^Y(\mathbf{x}_{Y'}) = \max_{Y \setminus Y'} \Phi_Y(\mathbf{x}_Y)$ {$\Theta(N^{|Y|})$}
 4: compute $\Psi^Z(\mathbf{x}_{Z'}) = \max_{Z \setminus Z'} \Phi_Z(\mathbf{x}_Z)$ {$\Theta(N^{|Z|})$}
 5: for $\mathbf{n} \in \mathrm{dom}(X \cap Y)$ do
 6:    compute $\mathbf{P}^Y[\mathbf{n}]$ by sorting $\Psi^Y(\mathbf{n}; \mathbf{x}_{Y' \setminus X})$
       {$\Psi^Y(\mathbf{n}; \mathbf{x}_{Y' \setminus X})$ is free over $\mathbf{x}_{Y' \setminus X}$; $\mathbf{P}^Y[\mathbf{n}]$ stores the $|Y' \setminus X|$-dimensional indices that sort it}
 7: end for {this loop takes $\Theta(|Y' \setminus X| N^{|Y'|} \log N)$}
 8: for $\mathbf{n} \in \mathrm{dom}(X \cap Z)$ do
 9:    compute $\mathbf{P}^Z[\mathbf{n}]$ by sorting $\Psi^Z(\mathbf{n}; \mathbf{x}_{Z' \setminus X})$
10: end for {this loop takes $\Theta(|Z' \setminus X| N^{|Z'|} \log N)$}
11: for $\mathbf{n} \in \mathrm{dom}(X')$ do
12:    $(\mathbf{v}_a, \mathbf{v}_b) = (\Psi^Y(\mathbf{n}|_{Y'}; \mathbf{x}_{Y' \setminus X'}), \Psi^Z(\mathbf{n}|_{Z'}; \mathbf{x}_{Z' \setminus X'}))$
       {$\mathbf{n}|_{Y'}$ is the 'restriction' of the vector $\mathbf{n}$ to those indices in $Y'$; hence $\Psi^Y(\mathbf{n}|_{Y'}; \mathbf{x}_{Y' \setminus X'})$ is free in $\mathbf{x}_{Y' \setminus X'}$, while $\mathbf{n}|_{Y'}$ is fixed}
13:    $(p_a, p_b) = (\mathbf{P}^Y[\mathbf{n}|_{Y'}], \mathbf{P}^Z[\mathbf{n}|_{Z'}])$
14:    $best = Algorithm1\ (\mathbf{v}_a, \mathbf{v}_b, p_a, p_b)$
15:    $m_X(\mathbf{n}) = \Psi^X(\mathbf{n})\Psi^Y(best; \mathbf{n}|_{Y'})\Psi^Z(best; \mathbf{n}|_{Z'})$
16: end for {this loop takes $O(N^{|X'|}\sqrt{N^{|(Y' \cap Z') \setminus X'|}})$}
17: $m_M(\mathbf{x}_M) = Naive(m_X, M)$ {i.e., we are using the naïve algorithm to marginalise $m_X(\mathbf{x}_X)$ with respect to $M$; this takes $\Theta(N^{|X|})$}
```

The marginalisation steps of Algorithm 3 (Lines 2, 3, and 4) may further decompose into smaller groups, in which case Algorithm 3 can be applied recursively. For instance, the graph in Figure 5(a) shows the marginalisation step from Algorithm 3, Line 4 (see Fig. 4(c)). Since this marginalisation step is the asymptotically dominant step in the algorithm, applying Algorithm 3 recursively lowers the asymptotic complexity.

Naturally, there are cases for which a decomposition into three terms is not possible, such as

$$m_{i,j,k}(x_i, x_j, x_k) = \max_{x_m} \Phi_{i,j,k}(x_i, x_j, x_k) \times \Phi_{i,j,m}(x_i, x_j, x_m)\Phi_{i,k,m}(x_i, x_k, x_m)\Phi_{j,k,m}(x_j, x_k, x_m) \quad (9)$$

(i.e., a clique of size four with third-order factors). However, if the model contains factors of size K, it must always be possible to split it into $K+1$ groups (e.g. four in the case of (eq. 9)).

Our optimizations can be applied in these cases simply by adapting Algorithm 1 to solve problems of the form

$$\hat{i} = \underset{i \in \{1 \dots N\}}{\mathrm{argmax}} \{\mathbf{v}_1[i] \times \mathbf{v}_2[i] \times \cdots \times \mathbf{v}_K[i]\}. \quad (10)$$

Figure 3(c) demonstrates how such an algorithm behaves in practice: if we have K lists, the cube in Figure 3(c) becomes a K-dimensional hypercube. Pseudocode is not shown, though it is similar to Algorithm 1. Again, we shall discuss the running time of this extension in Section 3.2. For the moment, we state the following theorem:

Theorem 3. *Algorithm 1 generalises to K lists with an expected running time of $O(KN^{\frac{K-1}{K}})$, yielding a speed-up of $\Omega(\frac{1}{K}N^{\frac{1}{K}})$ in cliques containing K-ary factors (it can be adapted to be $O(\min(N, KN^{\frac{K-1}{K}}))$, if we carefully avoid rereading entries).*

Using this extension, we can extend Algorithm 3 to allow for any number of *groups* (pseudocode is not shown; all statements about the groups Y and Z simply become statements about K groups $\{G_1 \dots G_K\}$). The one remaining case that has not been considered is when the sequences $\mathbf{v}_1 \cdots \mathbf{v}_K$ are functions of different (but overlapping) variables; this can be trivially circumvented by 'padding' each of them to be functions of the same variables, and by carefully applying recursion.

As a final comment we note that we have not provided an algorithm for choosing how to split the variables into $(K+1)$-groups, and we note that different splits may result in better performance. However, even if the split is chosen in a naïve way, we will still get the performance increases mentioned.

3.2 EXPECTED-CASE COMPLEXITY

In this section we shall determine the expected-case running time of Algorithm 1. Algorithm 1 traverses $\mathbf{v}_a$ and $\mathbf{v}_b$ until it reaches the smallest value of m for which there is some $j \leq m$ such that $m \geq p_b^{-1}[p_a[j]]$. Extensions of Algorithm 1 aim to find the smallest m for which

$$\max(i, p_1[i], \dots, p_{K-1}[i]) \leq m. \quad (11)$$

If M is a random variable representing this smallest value of m, then we wish to find $E(M)$. Simple analysis reveals that the probability of choosing a single permutation p such that $\max(i, p[i]) > m$ is

$$P(M > m) = \frac{(N-m)!(N-m)!}{(N-2m)!N!}. \quad (12)$$

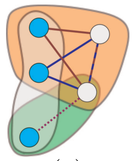
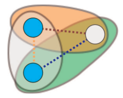
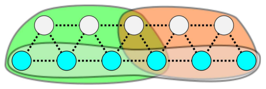
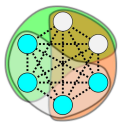
	(a)	(b)	(c)	(d)	(e)
Graph:					{The complete graph K_M, with pairwise terms}
Naïve solution:	$\Theta(N^5)$	$\Theta(N^3)$	$\Theta(N^{11})$	$\Theta(N^6)$	$\Theta(N^M)$
Algorithm 3:	$O(N^3\sqrt{N})$	$O(N^2\sqrt{N})$	$O(N^6\sqrt{N})$	$O(N^5)$	$O(N^{5M/6})$
Speed-up:	$\Omega(N\sqrt{N})$	$\Omega(\sqrt{N})$	$\Omega(N^4\sqrt{N})$	$\Omega(N)$	$\Omega(N^{M/6})$

Figure 5: Some example graphs whose max-marginals are to be computed with respect to the coloured nodes, using the three regions shown. Factors are indicated using differently coloured edges, while dotted edges always indicate pairwise factors. (a) is the region Z from Figure 4 (recursion is applied *again* to achieve this result); (b) is the graph used to motivate Algorithm 2; (c) shows a query in a graph with regular structure; (d) shows a complete graph with six nodes; (e) generalises this to a clique with M nodes.

This is precisely $1 - F(m)$, where $F(m)$ is the cumulative density function of M. It is immediately clear that $1 \leq M \leq \lfloor N/2 \rfloor + 1$, which defines the best and worst-case performance of Algorithm 1.

Using the identity $E(X) = \sum_{x=1}^{\infty} P(X \geq x)$, we can write down a formula for the expected value of M

$$E(M) = \sum_{m=0}^{\lfloor N/2 \rfloor} \frac{(N-m)!(N-m)!}{(N-2m)!N!}, \tag{13}$$

which reflects the expected running time of Algorithm 1. Unfortunately, the corresponding expectation when we have $K-1$ permutations is not trivial to compute. It is possible to write down a formula that generalizes (eq. 12), though the expression is complicated and not very informative; hence we shall instead rely on the upper bounds mentioned in Theorems 2 and 3. Proofs of these bounds are given in Appendix A.

4 EXISTING APPLICATIONS

Our results are immediately compatible with several applications that rely on inference in graphical models. As we have mentioned, our results apply to *any model whose cliques decompose into lower-order terms.*

Often, potentials are defined only on *nodes* and *edges* of a model. A D^{th}-order Markov model has a tree-width of D, but in some cases contains only pairwise relationships. Similarly 'skip-chain CRFs' [Sutton and McCallum, 2006, Galley, 2006], and Junction-Trees used in SLAM applications [Paskin, 2003] often contain only pairwise terms. In each case, if the tree-width is D, Algorithm 3 takes $O(MN^D\sqrt{N})$ (for a model with M nodes and N states per node), yielding a speed-up of $\Omega(\sqrt{N})$.

Models for shape matching often exhibit similar properties [Sigal and Black, 2006]. Third-order cliques factorise into second order terms, resulting in a speed-up of $\Omega(\sqrt{N})$. Another similar model for shape matching is that of [Felzenszwalb, 2005]; this model again contains third-order cliques, though it includes a 'geometric' term constraining all three variables. However, the third-order term is *independent of the input data*, meaning that each of its rows can be sorted *offline*. Here we have an instance of Algorithm 1 with three lists, yielding a speed-up of $\Omega(N^{\frac{1}{3}})$.

In [Coughlan and Ferreira, 2002], deformable shape-matching is solved approximately using Loopy Belief-Propagation. Their model has only second-order cliques, meaning that inference takes $\Theta(MN^2)$ *per iteration*. Although we cannot improve upon this result, we note that we can typically do *exact* inference in a single iteration in $O(MN^2\sqrt{N})$; thus our model has the same running time as $O(\sqrt{N})$ iterations of the original version. This result applies to all models containing a single loop.

In [McAuley et al., 2008], a model is presented for graph-matching using Loopy Belief-Propagation; the maximal cliques for D-dimensional matching have size $(D+1)$, meaning that inference takes $\Theta(MN^{D+1})$ *per iteration* (it is shown to converge to the correct solution); we improve this to $O(MN^D\sqrt{N})$.

Belief-propagation can be used to compute *LP-relaxations* in pairwise graphical models. In [Sontag et al., 2008], LP-relaxations are computed for pairwise models by constructing several third-order 'clusters', which compute pairwise messages for each of their edges.

Table 1 summarizes these results. Running times reflect the *expected case*, assuming that *max-product belief-propagation is used in a discrete model.* Some of the referenced articles may suggest variants of the algorithm (e.g. Gaussian models, or approximate schemes); we believe that our approach may revive the exact, discrete version as a tractable option in such cases.

Table 1: Some existing work to which our results can be immediately applied (M nodes, N states per node).

REFERENCE	APPLICATION	RUNNING TIME	OUR APPROACH
[McAuley et al., 2008]	D-d graph-matching	$\Theta(MN^{D+1})$, iterative	$O(MN^{D}\sqrt{N})$, iterative
[Sutton and McCallum, 2006]	Width-D skip-chain	$O(MN^{D+1})$	$O(MN^{D}\sqrt{N})$
[Paskin, 2003] (discrete case)	SLAM, width D	$O(MN^{D+1})$	$O(MN^{D}\sqrt{N})$
[Felzenszwalb, 2005]	Deformable matching	$\Theta(MN^{3})$	$\Theta(MN^{\frac{8}{3}})$ + offline steps
[Coughlan and Ferreira, 2002]	Deformable matching	$\Theta(MN^{2})$, iterative	$O(MN^{2}\sqrt{N})$
[Sigal and Black, 2006]	Pose reconstruction	$\Theta(MN^{3})$	$O(MN^{2}\sqrt{N})$
[Sontag et al., 2008]	LP with M clusters	$\Theta(MN^{3})$	$O(MN^{2}\sqrt{N})$

5 EXPERIMENTS

5.1 PERFORMANCE AND BOUNDS

For our first experiment, we compare the performance of Algorithm 1 (and extensions) to the naïve solution. This is a core subroutine of each of the other algorithms, meaning that determining its performance shall give us an indication of the improvements we expect to obtain in real graphical models.

For each experiment, we generate N i.i.d. samples from $[0, 1)$ to obtain the lists $v_1 \dots v_K$. N is the domain size; this may refer to a single node, or a *group* of nodes; thus large values of N may appear even for binary-valued models. K is the number of lists in (eq. 10); we can observe this number of lists only if we are working in cliques of size $K+1$, and then only if the factors are of size K; therefore smaller values of K are probably more realistic in practice (indeed, all but one of the applications in Section 4 had $K = 2$).

The performance of our algorithm is shown in Figure 6 (left), for $K \in \{2, 3, 4\}$. The performance reported is just the number of elements read from the lists. This is compared to N itself, which is the number of elements read by the naïve version. The upper-bounds from Section A are also reported, while the expected performance (i.e., (eq. 13)) is reported for $K = 2$ (we are not aware of an efficiently computable generalisation of (eq. 13) for $K > 2$).

5.2 CORRELATED VARIABLES

The expected case running time of our algorithm was obtained under the assumption that the lists had independent order-statistics, as was the case for the previous experiment. We suggested that we will obtain worse performance in the case of negatively correlated variables, and better performance in the case of positively correlated variables; we will assess these claims in this experiment.

We report the performance for two lists (i.e., for Algorithm 1), whose values are sampled from a 2-dimensional Gaussian, with covariance matrix

$$\Sigma = \begin{bmatrix} 1 & c \\ c & 1 \end{bmatrix}, \qquad (14)$$

meaning that the two lists are correlated with correlation coefficient c. Performance is shown in Figure 6 (centre) for different values of c.

5.3 2-D GRAPH MATCHING

Naturally, Algorithm 3 has additional overhead compared to the naïve solution, meaning that it will not be beneficial for small N. We reproduce the model from [McAuley et al., 2008], which performs 2-dimensional graph matching, using a loopy graph with cliques of size three, containing only second order potentials (as described in Section 4); the $\Theta(NM^3)$ performance of their method is reportedly state-of-the-art.

We perform matching between a *template* graph with M nodes, and a *target* graph with N nodes, which requires a graphical model with M nodes and N states per node (see [McAuley et al., 2008]). We fix $M = 5$ and vary N. Performance is shown in Figure 6 (right). The running times appear to be comparable after only $N \simeq 6$, meaning that our algorithm has a speed-up over the solution of [McAuley et al., 2008] of about $\frac{2}{5}\sqrt{N}$; thus it is significantly faster than the state-of-the-art solution, even for small values of N. Plots of $t = \frac{N^3}{4000}$ and $t/\frac{2\sqrt{N}}{5}$ are overlayed on Figure 6 (right) to estimate the runtime in seconds as a function of N.

6 CONCLUSION

We have presented a series of approaches that allow us to improve the performance of the Junction-Tree Algorithm for models that factorize into terms smaller than their maximal cliques. We are able to improve the expected computational complexity in models whose cliques factorize, no matter the size or number of factors. Our results increase the class of models for which exact inference remains a tractable option.

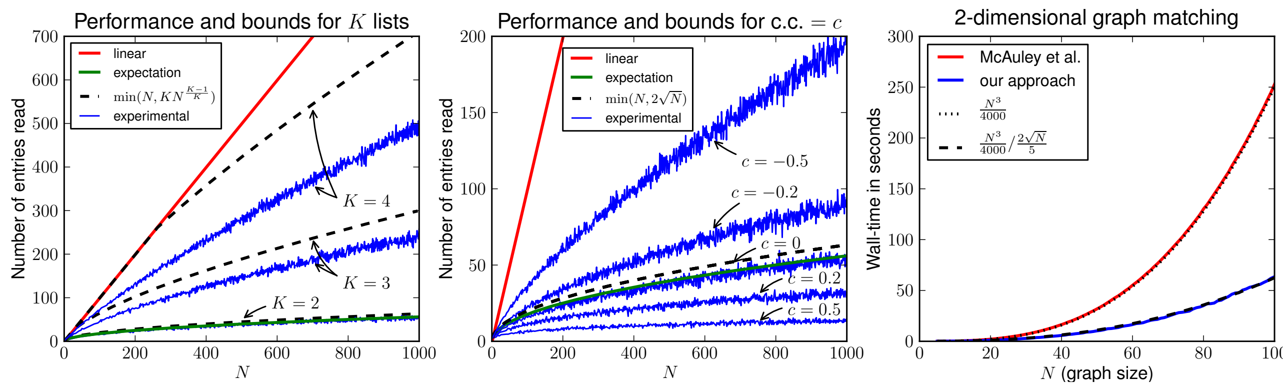


Figure 6: Left: Performance of our algorithm over 100 trials; the dotted lines show the bounds from Section A. Centre: Performance of our algorithm for different correlation coefficients. Right: The running time of our method on a graph matching experiment over 10 trials.

References

[Aji and McEliece, 2000] Aji, S. M. and McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343.

[Alon et al., 1997] Alon, N., Galil, Z., and Margalit, O. (1997). On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262.

[Coughlan and Ferreira, 2002] Coughlan, J. M. and Ferreira, S. J. (2002). Finding deformable shapes using loopy belief propagation. In *ECCV*.

[Felzenszwalb, 2005] Felzenszwalb, P. F. (2005). Representation and detection of deformable shapes. *IEEE Trans. on PAMI*, 27(2):208–220.

[Galley, 2006] Galley, M. (2006). A skip-chain conditional random field for ranking meeting utterances by importance. In *EMNLP*.

[Kschischang et al., 2001] Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519.

[McAuley et al., 2008] McAuley, J. J., Caetano, T. S., and Barbosa, M. S. (2008). Graph rigidity, cyclic belief propagation and point pattern matching. *IEEE Trans. on PAMI*, 30(11):2047–2054.

[Park and Darwiche, 2003] Park, J. D. and Darwiche, A. (2003). A differential semantics for jointree algorithms. In *NIPS*.

[Paskin, 2003] Paskin, M. A. (2003). Thin junction tree filters for simultaneous localization and mapping. In *IJCAI*.

[Sigal and Black, 2006] Sigal, L. and Black, M. J. (2006). Predicting 3d people from 2d pictures. In *AMDO*.

[Sontag et al., 2008] Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T., and Weiss, Y. (2008). Tightening LP relaxations for MAP using message passing. In *UAI*.

[Sutton and McCallum, 2006] Sutton, C. and McCallum, A. (2006). *An Introduction to Conditional Random Fields for Relational Learning.*

A BOUNDS AND PROOFS

Proof of Theorem 3 (sketch). We wish to determine the expected value of the smallest m satisfying (eq. 11). It can be shown that replacing the *permutations* in (eq. 11) with *random samples* of the values from 1 to N gives an *upper bound* on the expected value. This allows us to compute an upper bound on (eq. 12):

$$P(M > m) \leq \left(1 - \frac{m}{N}\right)^m, \tag{15}$$

and the corresponding version for K lists:

$$P^K(M > m) \leq \left(1 - \frac{m^{K-1}}{N^{K-1}}\right)^m. \tag{16}$$

In order to claim that the $E(M)$ is $O(f(N,K))$, (for K lists with N elements) it is sufficient to show that

$$\sum_{m=0}^{\infty} \left(1 - \frac{f(N,K)^{K-1}}{N^{K-1}}\right)^m \in O(f(N,K)). \tag{17}$$

Evaluating this geometric progression, we see that $f(N,K) = N^{\frac{K-1}{K}}$ is a suitable choice. At each step, K entries are read, resulting in the $O(KN^{\frac{K-1}{K}})$ time reported. $\square$

Theorem 2 is trivially proved as a special case of Theorem 3. To summarize, the expected running time of Algorithm 1 (for which we have $K = 2$ lists) is $O(\sqrt{N})$. This algorithm can be extended to handle K lists, with running time $O(KN^{\frac{K-1}{K}})$.

Acknowledgements

We would like to thank Pedro Felzenszwalb and Johnicholas Hines for comments on the first draft. NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative, and the Australian Research Council's *ICT Centre of Excellence* program.