Fast Inference with Min-Sum Matrix Product

Pedro F. Felzenszwalb
University of Chicago
pff@cs.uchicago.edu

Julian J. McAuley
Australian National University/NICTA
julian.mcauley@nicta.com.au

August 30, 2010

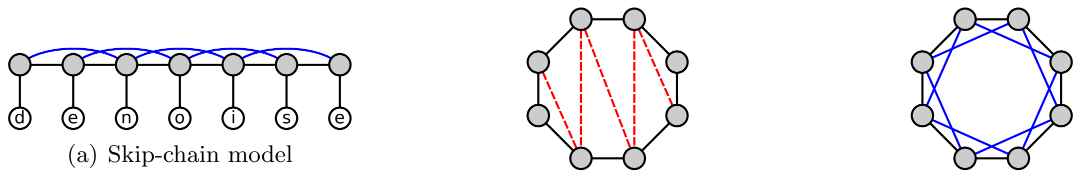
Abstract

The MAP inference problem in many graphical models can be solved efficiently using a fast algorithm for computing min-sum products of $n \times n$ matrices. Although the worst-case complexity of the min-sum product operation is not known to be much better than $O(n^3)$, an $O(n^{2.5})$ *expected time* algorithm was recently given, subject to some constraints on the input matrices. In this paper we give an algorithm that runs in $O(n^2 \log n)$ expected time, assuming the entries in the input matrices are independent samples from a uniform distribution. We also show that two variants of our algorithm are quite fast for inputs that arise in several applications. This leads to significant performance gains over previous inference methods in applications within computer vision and natural language processing.

1 Introduction

Min-sum matrix product (a.k.a. distance matrix product) is an important operation with applications in a variety of areas, including in inference algorithms for graphical models [12], parsing with context-free grammars [20], and shortest paths algorithms [1].

Our main interest in min-sum matrix product involves its application to MAP inference in graphical models. It is well known that inference in discrete graphical models with low tree-width can be done using dynamic programming and belief propagation. [12] showed that faster inference can be done in a large class of models if we have a fast method for computing min-sum matrix product. This class includes cyclic and skip-chain graphical models that arise in many applications including in natural language understanding and computer vision. See Figure 1 for some examples.



(a) Skip-chain model

(b) Triangulated cycle

(c) Point-matching model

Figure 1: Some typical graphical models with third-order cliques but only pairwise factors.

Let A and B be two $n \times n$ matrices. The min-sum product (MSP) of A and B is the $n \times n$ matrix $C = A \otimes B$ defined by

$$C_{ik} = \min_j A_{ij} + B_{jk}; \tag{1}$$

this is exactly matrix product in the min-plus (tropical) semiring.

Standard algorithms for MAP estimation with a tree-width 2 model take $O(mn^3)$ time, where m is the number of variables in the model and n is the number of possible values for each variable. For models that contain only pairwise factors inference can be done in $O(mf(n))$ time if we have an algorithm for computing MSP of $n \times n$ matrices in $O(f(n))$ time.

The brute-force approach for computing MSP of $n \times n$ matrices takes $O(n^3)$ time. Unfortunately there is no known method that improves this bound by a significant amount in the worst case. An important difference from the standard matrix product is that the minimum operation does not have an inverse. This means that fast matrix multiplication methods that rely on a ring structure, such as Strassen's algorithm [18], can not be directly applied to compute MSP.

Our main result is an algorithm for MSP that runs in $O(n^2 \log n)$ expected time, assuming the entries of each matrix are independent samples from a uniform distribution. Our basic algorithm uses a Fibonacci heap (or similar structure) and is mainly of theoretical interest. The algorithm can also be implemented with an integer queue to obtain a practical solution up to an additive error. We also describe an alternative algorithm that computes exact values using a scaling technique and avoids any complex data structure. Our experimental results show the methods perform well in three different applications: interactive image segmentation with active contours, point pattern matching with belief propagation and text denoising with skip-chain models.

1.1 Related Work

Our work is motivated by [12] who noted the application of MSP for MAP inference with graphical models and gave an $O(n^{2.5})$ expected time algorithm for MSP. The method in [12] assumes every permutation of values in the inputs occurs with equal probability. This is a weaker assumption than the one we make in our analysis and could lead to a faster method in some applications. Section 4 compares the two methods and shows our algorithms performs better in several applications.

The worst case complexity of the MSP operation has been heavily studied in the theoretical computer science community, especially because of its relation to the all-pairs-shortest-paths (APSP) problem. The asymptotic complexity of computing MSP of two $n \times n$ matrices is known to be the same as solving the APSP problem on a dense graph with n nodes [1]. To our knowledge the best algorithm for the APSP problem takes $O(n^3/\log n)$ time in the worst case [5]. The search for a truly sub-cubic algorithm ($O(n^{3-\epsilon})$) is a significant open problem in the area.

There are several known algorithms for the APSP problem which have good expected runtime assuming the input graph comes from a simple distribution (e.g. [8, 14, 9]). However, straightforward application of these algorithms for computing MSP does not lead to good running times. Most methods assume the graph is complete with i.i.d. weights while some methods allow for more general distributions. The reduction of MSP to APSP leads to graphs that do not fit these assumptions.

Our basic algorithm can be seen as an application of Knuth's lightest derivation algorithm (KLD) [11, 7] to the MSP problem with a special stopping criterion. [7] suggested the use of KLD and an A* version of it for inference in graphical models. However there is a difference between the approach we use here and the one suggested by [7]. When doing inference on a large model we solve several small lightest derivation problems (each defined by a single MSP computation). In contrast, [7] suggests solving a single large lightest derivation problem. Solving a sequence of small problems leads to better running time bounds and simplifies the implementation.

2 MAP Inference and Min-Sum Matrix Product

MAP inference in a graphical model consists of solving a problem of the form

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{C \in \mathcal{C}} \Phi_C(\mathbf{x}_C),$$

where $\mathcal{C}$ is the set of maximal cliques in the model, and Φ_C is a *potential function* over the nodes in C. Exact or approximate solutions are often found using some form of message passing technique. This includes classical dynamic programming methods [4], loopy belief propagation [21] and the junction-tree algorithm [2]. Sometimes messages are computed between cliques of the original graphical model and sometimes over a triangulated version of the model.

In general the message passed from a clique A to a clique B takes the form

$$m_{A \to B}(\mathbf{x}_{A \cap B}) = \min_{\mathbf{x}_{A \setminus B}} \left(\Phi_X(\mathbf{x}_A) + \sum_{C \in \Gamma(A) \setminus B} m_{C \to A}(\mathbf{x}_{A \cap C}) \right),$$

where $\Gamma(A)$ are the cliques neighboring A.

If the model is triangulated and $m_{A \to B}$ is computed after A receives messages from all neighbors except B (i.e., $\Gamma(A) \setminus B$) this leads to the junction-tree algorithm. In loopy belief propagation messages are updated in parallel or some arbitrary order until convergence. After messages are computed a solution can be obtained by computing beliefs in a similar fashion.

As noted in [12], there are many graphical models whose potential functions Φ_C are decomposable into smaller factors, i.e.,

$$\Phi_C(\mathbf{x}_C) = \sum_{F \subset C} \Phi_F(\mathbf{x}_F).$$

This is a general phenomenon that arises for example when one triangulates a model. Triangulation creates new edges, and thus larger cliques, but the potential functions of the triangulated graphs can always be decomposed into the original potential functions. As in [12], we focus on the case where the potentials take the form

$$\Phi_{ijk}(x_i, x_j, x_k) = \Phi_{ij}(x_i, x_j) + \Phi_{ik}(x_i, x_k) + \Phi_{jk}(x_j, x_k).$$

That is, we have cliques of size three with pairwise factors. Our algorithms can also be generalized to other factorizations discussed in [12], but we concentrate on this particular case because it is the most common in typical applications.

For example, if we have a cyclic model, globally optimal solutions $\mathbf{x}^*$ can be obtained by applying the junction-tree algorithm to a triangulated graph (Figure 1(b)). We describe experiments with a model of this type for image segmentation in Section 4.1. Another example is a skip-chain model [19] where we have a sequence of hidden variables and a potential function between pairs of variables that have distance at most two from each other (Figure 1(a)). Section 4.3 illustrates an application of a model of this type for text denoising.

In such cases, a message from a clique $A = \{i, j, k\}$ to a clique $B = \{i, l, k\}$ takes the form

$$m_{A \to B}(x_i, x_k) = \Psi_{ik}(x_i, x_k) + \min_{x_j} \Psi_{ij}(x_i, x_j) + \Psi_{jk}(x_j, x_k), \tag{2}$$

where Ψ_{ij} is the sum of Φ_{ij} and messages from cliques that intersect A at (i, j), Ψ_{jk} is the sum of Φ_{jk} and messages from cliques that intersect A at (j, k), and Ψ_{ik} is the sum of Φ_{ik} and messages from cliques, other than B, that intersect A at (i, k).

Note that (eq. 2) is essentially equivalent to MSP (eq. 1) of two matrices (Ψ_{ij} and Ψ_{jk}) of size n, where n is the number of possible values for each variable in the model. The only difference is that (eq. 2) requires adding another matrix (Ψ_{ik}) to the result. Suppose we can compute MSP of two $n \times n$ matrices in $O(f(n))$ time. Then we can compute messages in $O(f(n))$ time.

Consider a graphical model with m variables and where each variable can take one of n possible values. If the graph has tree-width 2 we can triangulate it and use the junction-tree algorithm to find an optimal solution to the MAP inference problem in $O(mf(n))$ time.

Like MSP can be used for MAP inference, standard matrix multiplication can be used for computing marginals. Thus matrix multiplication algorithms such as Strassen's method [18] can be used for marginal computation in the class of models we consider here. We note however that such methods are not very practical due to high constants.

3 The Algorithm

Here we describe our basic algorithm (Algorithm 1) for computing $C = A \otimes B$. We assume all entries in A and B are non-negative. Negative (finite) entries can be eliminated by adding a constant to each matrix and subtracting the constants from the resulting C.

Pseudocode for Algorithm 1 is shown in the left column of Figure 2. Initially we set $C_{ik} = \infty$ and insert all entries of A, B and C in a priority queue Q, with priority given by their value. We then repeatedly remove items from Q and insert them in S. Whenever A_{ij} (B_{jk}) is removed from Q we combine it with entries of the form B_{jk} (A_{ij}) that are in S and update $C_{ik} = \min(C_{ik}, A_{ij} + B_{jk})$. The algorithm terminates when all entries of C are in S.

Theorem 1 *If all entries in A and B are non-negative then Algorithm 1 correctly computes C.*

Proof: Let $j = \operatorname{argmin}_j A_{ij} + B_{jk}$. Clearly we always have $C_{ik} \geq A_{ij} + B_{jk}$. It suffices to show that when C_{ik} is removed from Q we have $C_{ik} = A_{ij} + B_{jk}$. Since the entries in A and B are non-negative $A_{ij}, B_{jk} \leq C_{ik}$ and both A_{ij} and B_{jk} will be removed from Q before C_{ik}. This implies that when C_{ik} is removed from Q we have $C_{ik} = A_{ij} + B_{jk}$. □

Theorem 2 *If all entries in A and B are i.i.d. samples from a uniform distribution then Algorithm 1 can be implemented to run in $O(n^2 \log n)$ expected time.*

Proof: First note that we can assume the entries in A and B come from a uniform distribution over $[0, 1]$ by scaling them and then re-scaling the resulting C accordingly.

In practice we can keep two arrays of linked lists I and K such that $I[j]$ stores indices i for which A_{ij} is in S while $K[j]$ stores indices k for which B_{jk} is in S. Then, when an entry is removed from Q we can find the entries in S that combine with it in constant time per entry. For example, when A_{ij} is removed from Q we iterate over k in $K[j]$. Thus the running time of the algorithm is dominated by the additions and priority queue operations.

Let N be the number of additions done by the algorithm. We perform $O(n^2)$ insertions and remove-min operations, and $O(N)$ decrease-key operations. Lemma 1 shows $E[N]$ is $O(n^2 \log n)$. Using a Fibonacci heap we obtain $O(1)$ insertion and decrease-key, and $O(\log n)$ remove-min. This leads to the running time bound of $O(n^2 \log n)$. □

Lemma 1 *Let N be the number additions performed by Algorithm 1. If the entries in A and B are i.i.d. samples from the uniform distribution over $[0, 1]$ then $E[N]$ is $O(n^2 \log n)$*

Proof: Let $C = A \otimes B$, and M be the maximum value in C. The algorithm only adds A_{ij} and B_{jk} if both are at most M. Otherwise at least one of A_{ij} or B_{jk} will not be removed from Q before the algorithm stops. Let $X_{ijk} = 1$ if $A_{ij} \leq M$ and $B_{jk} \leq M$, and 0 otherwise. The number of additions performed by the algorithm is $N = \sum_{ijk} X_{ijk}$.

Using linearity of expectation we have

$$E[N] = \sum_{ijk} E[X_{ijk}] = \sum_{ijk} P(X_{ijk} = 1).$$

First we show that M is small with high probability because each entry in C is the minimum of n values. Then we use the fact that A_{ij} and B_{jk} are both small with low probability. This will imply that $X_{ijk} = 1$ with very low probability (diminishing with n).

Algorithm 1 Find $C_{ik} = \min_j A_{ij} + B_{jk}$

```
1:  $S := \varnothing$
2:  $C_{ik} := \infty$
3:  Initialize $Q$ with entries of $A$, $B$, $C$
4:  while $S$ does not contain all $C_{ik}$ do
5:      $item$ := remove-min($Q$)
6:      $S := S \cup item$
7:      if $item = A_{ij}$ then
8:          for $B_{jk} \in S$ do
9:              if $A_{ij} + B_{jk} < C_{ik}$ then
10:                 $C_{ik} := A_{ij} + B_{jk}$
11:                 decrease-key($Q, C_{ik}$)
12:             end if
13:         end for
14:     end if
15:     if $item = B_{jk}$ then
16:         for $A_{ij} \in S$ do
17:             if $A_{ij} + B_{jk} < C_{ik}$ then
18:                 $C_{ik} := A_{ij} + B_{jk}$
19:                 decrease-key($Q, C_{ik}$)
20:             end if
21:         end for
22:     end if
23: end while
```

Algorithm 2 Find $C_{ik} = \min_j A_{ij} + B_{jk}$

```
1:  $C_{ik} := \infty$
2:  $T := t\text{-}min$
3:  while $\max_{ik} C_{ik} > T$ do
4:      $I[j] := \{i \mid A_{ij} \leq T\}$
5:      $K[j] := \{k \mid B_{jk} \leq T\}$
6:      for $j \in \{1 \ldots n\}$ do
7:          for $i \in I[j]$ do
8:              for $k \in K[j]$ do
9:                  $c := A_{ij} + B_{jk}$
10:                 if $c < C_{ik}$ then
11:                     $C_{ik} := c$
12:                 end if
13:             end for
14:         end for
15:     end for
16:     $T := 2T$
17: end while
```

Figure 2: Two algorithms for computing MSP. The first (left) uses a Fibonacci heap or similar data structure. The second (right) avoids the use of 'exotic' data structures and is very fast in practice but may be sensitive to the schedule for T.

Let ϵ be a value between 0 and 1. We have that $M \geq \epsilon$ iff some $C_{ik} \geq \epsilon$. Using the union bound

$$P(M \geq \epsilon) \leq \sum_{ik} P(C_{ik} \geq \epsilon).$$

$C_{ik} \geq \epsilon$ iff for all j we have $A_{ij} + B_{jk} \geq \epsilon$. For fixed (i, k) these are independent events and thus

$$P(C_{ik} \geq \epsilon) = \prod_{j} P(A_{ij} + B_{jk} \geq \epsilon).$$

Let $s = A_{ij} + B_{jk}$. Since A_{ij} and B_{jk} are independent samples from the uniform distribution over $[0, 1]$, we have that $P(s = x)$ is the convolution of two boxes, and equals a triangle. For $0 \leq x \leq 1$ we have $P(s = x) = x$, and $P(s \leq x) = x^2/2$. Thus

$$P(A_{ij} + B_{jk} \geq \epsilon) = 1 - \epsilon^2/2 \quad \text{and} \quad P(C_{ik} \geq \epsilon) = (1 - \epsilon^2/2)^n.$$

Using $1 - x \leq e^{-x}$ we obtain

$$P(C_{ik} \geq \epsilon) \leq e^{-n\epsilon^2/2}.$$

Now we can see that M is small with high probability, or large with low probability

$$P(M \geq \epsilon) \leq n^2 e^{-n\epsilon^2/2}.$$

Note that $P(A_{ij} \leq \epsilon) = P(B_{jk} \leq \epsilon) = \epsilon$. Since these are independent events we have

$$P(A_{ij} \leq \epsilon \wedge B_{jk} \leq \epsilon) = \epsilon^2. \tag{3}$$

Let E_1 denote the event that $M \geq \epsilon$ and E_2 denote the event that $A_{ij} \leq \epsilon \wedge B_{jk} \leq \epsilon$. We have that $X_{ijk} = 1$ requires at least one of E_1 or E_2 to hold. Using the union bound

$$P(X_{ijk} = 1) \leq P(E_1) + P(E_2) \leq n^2 e^{-n\epsilon^2/2} + \epsilon^2.$$

Now we can pick ϵ so that both terms above are small. It is sufficient to pick $\epsilon^2 = \frac{6 \log n}{n}$. Note that as long as n is big enough we satisfy the requirement that $\epsilon \leq 1$. With this choice

$$P(X_{ijk} = 1) \leq \frac{1 + 6 \log n}{n}.$$

Finally we obtain

$$E[N] \leq n^3 \left(\frac{1 + 6 \log n}{n} \right) = n^2 (1 + 6 \log n). \tag{4}$$

So $E[N]$ is $O(n^2 \log n)$. $\square$

3.1 Approximate Solutions with an Integer Queue

To obtain the desired runtime bound for Algorithm 1 we need a complex data structure, such as a Fibonacci heap, supporting $O(1)$ time decrease-key operations. We have found that this leads to poor performance in practice since such data structures are relatively slow.

Suppose the entries in A and B are integers in $[0, K]$. Then we can initialize C_{ik} to $2K$, and the priorities in Q will always be integers in $[0, 2K]$. We can represent such a queue by an array of length $2K + 1$, with entry $Q[p]$ holding a list of values with priority p.

An important property of Algorithm 1 is that the minimum priority of items in Q never decreases. This is because when the value of C_{ik} is decreased, it does not go below the last value removed from Q. This makes it possible to perform k queue operations in $O(k + K)$ time. Initialization takes $O(K)$ time. Insertions and decrease-key each take $O(1)$ time, while k remove-min operations take $O(k + K)$ total. During remove-min we may need to search for the minimum p with $Q[p]$ not empty. But since the minimum never decreases we never need to search over the same priority twice.

Using an integer queue Algorithm 1 runs in $O(n^2 \log n + K)$ time. If the entries in A and B are not integers or have high value, we can scale and round them to ensure K is not too large. This approach introduces an additive error bounded by the size of the priority bins in Q. In particular, if the maximum value in A and B is v, we compute C_{ik} to within an additive error of v/K.

3.2 Scaling Method

Here we describe an alternative algorithm (Algorithm 2) that avoids using a priority queue and computes exact solutions to the MSP problem. Pseudocode for Algorithm 2 is shown in the right column of Figure 2.

Note that if we knew $M = \max_{ik} C_{ik}$, a very simple algorithm could be developed for computing C. Since the entries in A and B are non-negative, we have that $C_{ik} = \min_j A_{ij} + B_{jk}$ for those j such that $A_{ij} \leq M \wedge B_{jk} \leq M$. By (eq. 4) this would allow us to compute C in $O(n^2 \log n)$ expected time, without any need for a priority queue.

Of course we do not know M in advance. In Algorithm 2 we guess a value T and compute $C_{ik} = \min_j A_{ij} + B_{jk}$ for those j such that $A_{ij} \leq T \wedge B_{jk} \leq T$. If $\max_{ik} C_{ik} \leq T$ we have correctly computed C since larger values of A_{ij} or B_{jk} could not lead to smaller values for C_{ik}. Otherwise we double T and try again.

If we were able to set $T := M$ in the beginning of Algorithm 2 it would perform the same additions as Algorithm 1. Algorithm 2 terminates before $T > 2M$. By (eq. 3), doubling T increases the expected number of additions by a factor of 4. Thus the total expected number of additions performed by Algorithm 2 is at most a constant times the number of additions performed by Algorithm 1. In each iteration Algorithm 2 also takes $O(n^2)$ time to check C and initialize the lists I and K. The total expected runtime is $O(n^2 \log n)$ as long as the number of iterations is $O(\log n)$. This holds as long as the initial value for T is not too small.

3.3 Speedups

There are several speedups that improve the running time of our algorithms in practical situations:

1) From each entry in A we subtract the minimum value in its row, and from each entry in B we subtract the minimum value in its column. These minima are added back to the resulting C. This makes the values in C closer to the values in A and B and allows the algorithm to stop earlier.

2) In both algorithms we can remove entries from the set S, or lists I and K, if we already know all values in a row or column of C. We keep track of which rows/columns of C are done, and remove the entries the first time we consider them and realize they can no longer affect the result.

3) Let $a(j) = \min_i A_{ij}$ and $b(j) = \min_k B_{jk}$ be column and row minima in A and B. Entries in C derived from A_{ij} must have value at least $A_{ij} + b(j)$, and entries in C derived from B_{jk} must have value at least $a(j) + B_{jk}$. This can be used to increase the priority of the initial items in Algorithm 1, and to define an adaptive threshold for entries to include in I and K in Algorithm 2. This leads to A^* versions of our algorithms (the idea follows from the A* version of KLD [7]) and is beneficial in practice, for example in situations where the matrices A and B have different scales.

4 Experiments

We implemented our algorithms and tested them in several applications by comparing to the naïve (brute force) method for MSP and the method from [12]. Our implementation of Algorithm 1 uses an integer queue, as described in Section 3.1. Both Algorithms 1 and 2 were implemented using all speedups described in Section 3.3. All methods were implemented in C++ using the GNU C compiler on an 2.8Ghz Intel PowerMac running Mac OS X 10.5.

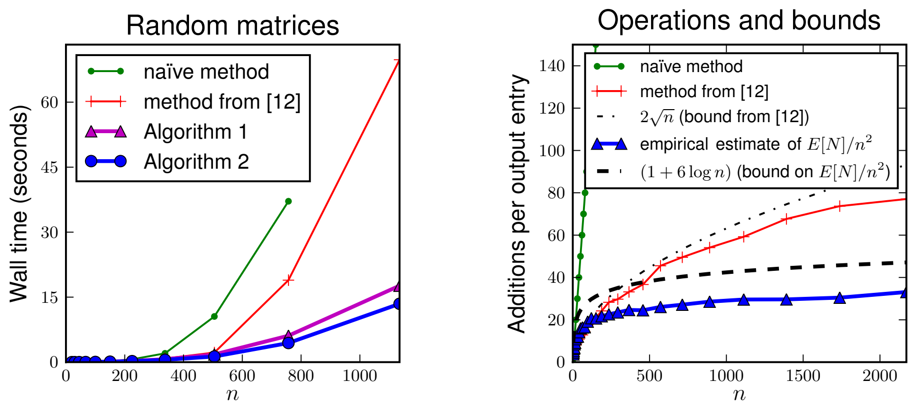


Figure 3: Left: runtime of different MSP algorithms. Right: number of additions per output entry.

First we evaluate our algorithms on uniform i.i.d. matrices. Note that such matrices satisfy the random order statistics assumption made in [12]. Figure 3 shows the performance of the different methods with running times on the left, and the number of additions done by each method on the right. The experiment confirms our methods have better asymptotic complexity on random inputs.

The experiments below show that results on structured inputs that arise in practical applications are similar to the random case.

4.1 Interactive Image Segmentation

Here we consider the problem of image segmentation using active contour models ('snakes') [10, 3]. Figure 4 illustrates an example. In this application a coarse segmentation of an object is provided by the user, in the form of a polygonal curve with m control points. The goal is to improve the segmentation by moving the control points within a search window around their initial positions.

In our experiments the final segmentation was obtained by solving a problem of the form

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{i=1}^{m} \frac{1}{grad(x_i, x_{i+1})} + ||x_i - x_{i+1}||^2.$$

Here x_i is constrained to be in a $w \times w$ window around the i-th control point. The value $grad(p, q)$ is a measure of the gradient magnitude along the line segment between p and q, (we want the object boundary to align with high gradient regions). The second term in the sum encourages compact boundaries with control points that are approximately uniformily spaced.

Solving for $\mathbf{x}^*$ is equivalent to MAP estimation with a cyclic graphical model and can be done via the junction-tree algorithm in a triangulated graph, such as the one in Figure 1(b). The

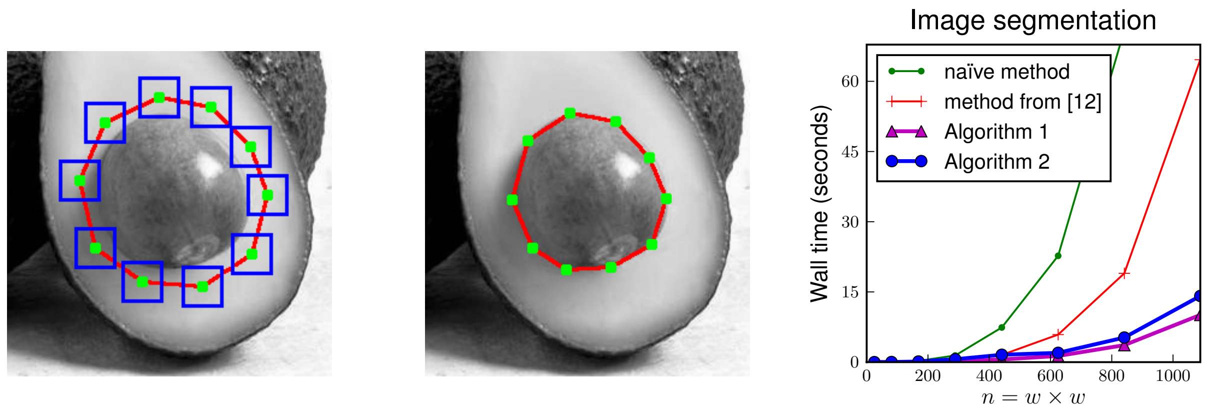


Figure 4: Interactive image segmentation with an active contour model. Left: initial placement of the contour and search neighborhoods for the control points. Center: final segmentation. Right: running time as a function of the search space size using different MSP algorithms.

standard inference procedure requires $O(mn^3)$, where $n = w^2$ is the number of possible positions for each control point. Figure 4 shows a typical result and running times obtained using different methods for MSP as a subroutine. Using the naïve method for MSP is computationally equivalent to classical dynamic programming solutions for this problem.

4.2 Point Pattern Matching

Many of the problems suggested in [12] involved finding maps between two point sets. Examples include OCR [6], pose reconstruction [17], SLAM [15], and point pattern matching [13].

Here we search for a 'template' $\mathbf{s}$ with m points within a 'target' $\mathbf{t}$ containing n points. The target consists of a (transformed) copy of the template, together with noise and outliers. An example is shown in Figure 5. The objective function in question takes the form

$$f^* = \operatorname*{argmin}_{f} \sum_{(i,j) \in E} g(||s_i - s_j||, ||f(s_i) - f(s_j)||),$$

where f maps points in $\mathbf{s}$ to points in $\mathbf{t}$ and for $(i,j) \in E$ we have a robust elasticity constraint defined by g, enforcing distances to be preserved to the extent possible.

Solving for f^* corresponds to MAP estimation in a graphical model with topology defined by E. It was shown in [12] that in many applications E forms a tractable model. We use the model from [13] shown in Figure 1(c). For inference we run loopy belief propagation for 25 iterations in

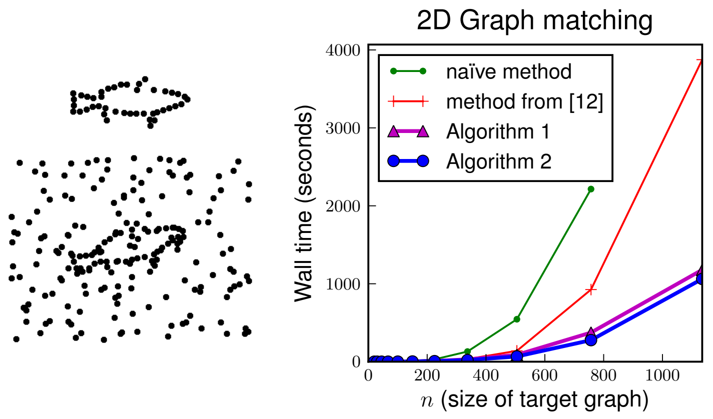


Figure 5: Point pattern matching experiment. Left: template and a scene with noise and outliers. Right: running times for inference using different algorithms.

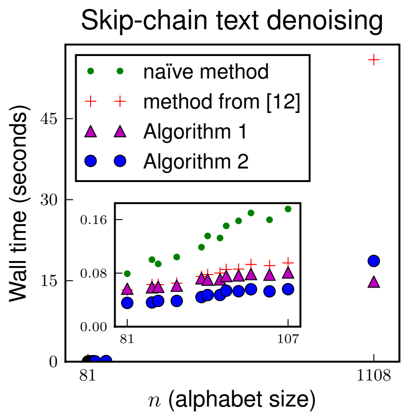


Figure 6: Text denoising experiment with different languages (box is closeup of bottom left).

the loop of 'width' 2. This takes $O(mn^3)$ time per iteration using the naïve MSP method. The performance using different MSP methods is shown in Figure 5.

4.3 Skip-Chain Models for Text Denoising

In [19], it was observed that powerful inference procedures can be developed by introducing long-range dependencies into pairwise graphical models.

In this experiment, we adapt a simple Markov model for text denoising (typo correction): we model not only the relationship between neighboring characters, but also the relationship between characters at distance two. This leads to the graphical model shown in Figure 1(a).

For a sequence of length m drawn from an alphabet with n characters the objective we use is

$$\mathbf{x}^*(\mathbf{t}) = \underset{\mathbf{x}}{\operatorname{argmax}} \underbrace{\prod_{i=1}^{m} \left[p(1 - I_{\{t_i\}}(x_i)) + (1-p) I_{\{t_i\}}(x_i)\right]}_{\text{noise model}} \underbrace{\prod_{i=1}^{m-1} q_1(x_i, x_{i+1}) \prod_{i=1}^{m-2} q_2(x_i, x_{i+2})}_{\text{prior}}.$$

Here $\mathbf{t}$ is our input text, with each character corrupted with probability p, and $I_A(x)$ is the indicator function that equals 1 if $x \in A$ and 0 if $x \notin A$. Our priors are extracted from the statistics of sentences in the Leipzig corpora [16]. Inference again requires $O(mn^3)$ operations using the naïve MSP method. The performance using different methods is shown in Figure 6. The largest language we consider is Korean with 1108 characters.

5 Conclusion

The MSP operation plays an important role for inference in a large class of graphical models. Our basic algorithm runs in $O(n^2 \log n)$ expected time assuming the entries in each input matrix are independent samples from a uniform distribution. Despite the strong assumption we show the algorithm can be made very fast for inputs that arise in practical applications, achieving significant performance gains over other methods. An interesting open question involves showing the algorithm has good running time bounds for more general inputs once we include the speedups described in Section 3.3. Another direction for future work involves other applications of MSP, such as parsing with context-free grammars.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

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

[3] Amir Amini, Terry Weymouth, and Ramesh Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, 1990.

[4] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

[5] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Annual ACM Symposium on Theory of Computing*, pages 590–598, 2007.

[6] James M. Coughlan and Sabino J. Ferreira. Finding deformable shapes using loopy belief propagation. In *ECCV*, 2002.

[7] Pedro F. Felzenszwalb and David McAllester. The generalized A* architecture. *Journal of Artificial Intelligence Research*, 29:153–190, 2007.

[8] A. M. Frieze and G. R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.

[9] David R. Karger, Daphne Koller, and Steven J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal of Computing*, 22(6):1199–1217, 1993.

[10] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[11] Donald Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, 1977.

[12] Julian J. McAuley and Tibério S. Caetano. Exploiting within-clique factorizations in junction-tree algorithms. In *AISTATS*, 2010.

[13] Julian J. McAuley, Tibério S. Caetano, and Marconi S. Barbosa. Graph rigidity, cyclic belief propagation and point pattern matching. *IEEE Transansactions on Pattern Analysis and Machine Intelligence*, 30(11):2047–2054, 2008.

[14] Alistair Moffat and Tadao Takaoka. An all pairs shortest path algorithm with expected time $O(n^2 \log n)$. *SIAM Journal of Computing*, 16(6):1023–1031, 1987.

[15] Mark A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *IJCAI*, 2003.

[16] U. Quasthoff, M. Richter, and C. Biemann. Corpus portal for search in monolingual corpora. In *Language Resources and Evaluation*, 2006.

[17] Leonid Sigal and Michael J. Black. Predicting 3D people from 2D pictures. In *AMDO*, 2006.

[18] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356, 1969.

[19] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. 2006.

[20] Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10:308–315, 1975.

[21] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, 2000.