CDoT: Optimizing MAP Queries on Trees

Roberto Esposito, Daniele P. Radicioni, and Alessia Visconti

Department of Computer Science, University of Torino
{roberto.esposito,daniele.radicioni,alessia.visconti}@unito.it

Abstract. Among the graph structures underlying Probabilistic Graphical Models, trees are valuable tools for modeling several interesting problems, such as linguistic parsing, phylogenetic analysis, and music harmony analysis. In this paper we introduce CDoT, a novel exact algorithm for answering Maximum a Posteriori queries on tree structures. We discuss its properties and study its asymptotic complexity; we also provide an empirical assessment of its performances, showing that the proposed algorithm substantially improves over a dynamic programming based competitor.

1 Introduction

In this paper we introduce CarpeDiem on Trees (CDoT), an algorithm for solving efficiently and exactly Maximum A Posteriori (MAP) queries on tree-structured probabilistic graphical models.

Probabilistic Graphical Models (PGMs) lie at the intersection of probability and graph theory; they sport a rigorous theoretical foundation and provide an abstract language for modeling application domains [1]. PGMs have been successfully applied in fields as diverse as medical diagnosis [2], computer vision [3], and text mining [4]. Informally stated, a PGM is a graph specifying the conditional independence structure of a family of probability distributions. Specifically, vertices are used to represent random variables, and edges are used to model an intervening direct probabilistic interaction between two random variables. The conditional independence structure is the key property used to efficiently handle the otherwise intractable problems of inference and learning.

Among the graph structures underlying PGMs, trees are valuable tools for modeling several interesting problems. They have been successfully applied to solve challenging tasks, such as linguistic parsing [5], phylogenetic analysis [6], and music harmony analysis [7].

Answering MAP queries over a PGM entails finding the assignment to the graph variables that *globally* maximizes the probability of an observation. This differs from optimizing the assignment to each variable in isolation. For instance, in the optical character recognition task, the labeling "*learning*" is probably to be preferred over the labeling "*1earnin9*" –based on known interactions between nearby labels–, even though the observations of the first and last characters might suggest otherwise when considered in isolation. MAP queries are important reasoning tools *per se*, but they are of the utmost importance during learning of

M. Baldoni et al. (Eds.): AI*IA 2013, LNAI 8249, pp. 481–492, 2013.
© Springer International Publishing Switzerland 2013

both the PGM structure and parameters. In fact, in tackling these two tasks, MAP queries are often used as repeatedly called sub-procedures.

The MAP problem is $\mathcal{NP}$-complete in general, but it can be solved in polynomial time on particular graph structures such as chains and trees by variations on the Viterbi algorithm [8]. Several algorithms have been proposed to efficiently solve MAP queries on chains. For instance, CarpeDiem [9] distinguishes between 'local' (associated with vertices) and 'transitional' (associated with edges) information to improve on the quadratic –in the number of labels– complexity of competing approaches. The main intuition therein contained is that local evidence often provides most of the clues needed to optimally solve the problem, while dependencies between variables can be used to discriminate between cases that cannot be differentiated otherwise. Recently, [10] exploited the same intuition to develop an algorithm based on linear programming techniques. In both cases experimental evidence shows impressive improvements with respect to the Viterbi algorithm.

In this paper we introduce CDoT, an algorithm that extends the approach in [9] to the case of tree structured PGMs. The algorithm always returns the optimal answer to the MAP query, often spending only a fraction of the computational resources demanded by competing approaches. We illustrate the algorithm in full details, provide a formal study of the algorithm complexity, and report on an experimentation comparing CDoT with a dynamic programming solution.

2 Formalization

Let us consider a probabilistic graphical model where a tree T encodes conditional independences, and P is the associated probability distribution. Let us assume, without loss of generality that T is a directed tree, and that it has edges oriented from root towards leaves. Each vertex V in T represents a random variable that takes values in the set of labels L_V; in the following, the notation l_V is used to denote a label in L_V. A MAP query over this structure amounts to determining the optimal assignment of labels to variables.[1] An optimal assignment is one that maximizes the cumulative reward of the root R of the tree.

A summary of the notation that we will be using throughout the paper is reported in Table 1.

We consider an execution of the Variable Elimination algorithm using any elimination ordering that eliminates leaves first and then moves toward the root R. The outcome is the evaluation of the expression:

$$\max_{l_R} \left[\sum_{V \in \mathrm{ch}(R)} \underbrace{\max_{l_V} \left[\phi(l_R, l_V) + \sum_{W \in \mathrm{ch}(V)} (\cdots) \right]}_{(a)} \right] \tag{1}$$

[1] If more than one optimal assignment exists for the tree, then any one of them is a valid answer to the MAP query.

Table 1. Summary of the adopted notation

Symbol	Description
$\|T\|$	the number of vertices in T
K	the number of labels per vertex
l_V	a label l in vertex V
λ	a distance from the tree root
$\omega(l_V)$	the cumulative reward up to label l_V
$\widetilde{\omega}\ (l_V)$	the estimated cumulative reward up to label l_V
$\mathcal{L}_{\sqsubseteq}(V)$	the sorted list of labels for vertex V
$\nu(l_V)$	a local factor
$\tau(l_V \twoheadrightarrow l_W)$	a transition factor
$\mathcal{T}$	the maximal transition weight between any two labels

where $\mathrm{ch}(V)$ is the set of children of V; $\phi(l_V, l_W)$ is a factor that depends only on the labels of variables V and W; and the dots substitute an expression analogous to expression (a). By using standard dynamic programming techniques Expression (1) can be evaluated in $O(|T|K^2)$ where $|T|$ is the number of vertices in T and K is the number of values (i.e., labels) that each variable can assume.

It is always possible to rewrite each $\phi(l_V, l_W)$ as the sum of a transition factor $\tau(l_V \twoheadrightarrow l_W)$ and a local factor $\nu(l_V)$, that is: of a factor $\nu(l_W)$ that only depends on labels of V, and of a factor $\tau(l_V \twoheadrightarrow l_W)$ that accounts for the relationship between the two vertices. In a directed model each factor $\phi(l_V, l_W)$ corresponds to the logarithm of the conditional probability $P(W = l_W|V = l_V)$. The proposed rewriting amounts to factorizing this probability as: $P(W = l_W|V = l_V) = \psi_0(l_W) \cdot \psi_1(l_V, l_W)$ with $\psi_0(l_W) > 0$, then setting $\nu(l_W) = \log(\psi_0(l_W))$ and $\tau(l_V \twoheadrightarrow l_W) = \log(\psi_1(l_V, l_W))$.

Given the above decomposition, we rewrite Expression (1) as:

$$\max_{l_R} \Big[\sum_{V \in \mathrm{ch}(R)} \max_{l_V} [\tau(l_R \twoheadrightarrow l_V) + \nu(l_V) + \sum_{W \in \mathrm{ch}(V)} (\cdots)] \Big].$$

By defining the cumulative reward $\omega(\cdot)$ as:

$$\omega(l_V) = \nu(l_V) + \sum_{W \in \mathrm{ch}(V)} \max_{l_W} [\tau(l_V \twoheadrightarrow l_W) + \omega(l_W)]$$

the expression can be rewritten as:

$$\max_{l_R} \left[\sum_{V \in \mathrm{ch}(R)} \max_{l_V} [\tau(l_R \twoheadrightarrow l_V) + \omega(l_V)] \right].$$

Finally, by defining $\nu(l_R)$ as 0, Expression (1) can be restated as $\max_{l_R} \omega(l_R)$.

CDoT computes expression $\max_{l_V} \omega(l_V)$ with sub-quadratic complexity. This complexity is achieved when the algorithm succeeds in avoiding the inspection of a number of labels by exploiting an upper bound to the cumulative reward.

Let $\mathcal{T}$ be an upper bound to the maximal transition weight between any two labels, that is: $\mathcal{T} \geq \max_{l_V, l_W} \tau(l_V \rightarrowtail l_W)$. Let $\sqsubseteq$ be a relation such that $l_V \sqsubseteq l'_V$ iff $\nu(l_V) \geq \nu(l'_V)$, and let $\mathcal{L}_{\sqsubseteq}(V)$ be the list of labels of vertex V, ordered according to $\sqsubseteq$. We say that a label l_V is more promising than label l'_V if $l_V \sqsubseteq l'_V$.

We denote by $\widetilde{\omega}\,(l_V)$ the upper bound for $\omega(l_V)$ defined as:

$$\widetilde{\omega}\,(l_V) = \nu(l_V) + |\mathrm{ch}(V)| \cdot \mathcal{T} + \sum_{W \in \mathrm{ch}(V)} \max_{l_W} \omega(l_W).$$

Two interesting properties of $\widetilde{\omega}\,(l_V)$ are:

$$\forall l_V, l'_V : \nu(l_V) \geq \nu(l'_V) \Leftrightarrow \widetilde{\omega}\,(l_V) \geq \widetilde{\omega}\,(l'_V) \tag{2}$$

$$\forall l_V : \qquad \widetilde{\omega}\,(l_V) \geq \omega(l_V) \tag{3}$$

Property (2) states that $\widetilde{\omega}\,(l_V)$ is monotone in $\nu(l_V)$. The property follows immediately by noticing that the quantity $|\mathrm{ch}(V)| \cdot \mathcal{T} + \sum_{W \in \mathrm{ch}(V)} \max_{l_W} \omega(l_W)$ for a fixed V does not depend on the actual label l_V. Property (3) states that $\widetilde{\omega}\,(l_V)$ is an upper bound for $\omega(l_V)$. It stems from *a)* noticing that $\widetilde{\omega}\,(l_V)$ can be obtained by substituting $\mathcal{T}$ in place of $\tau(l_V \rightarrowtail l_W)$ in the definition of $\omega(l_V)$; and *b)* recalling that by definition $\mathcal{T} \geq \tau(l_V \rightarrowtail l_W)$.

3 The Algorithm

In this section we illustrate the algorithm. We start by introducing the main ideas underlying the optimization strategy, and then we delve into the details of CDoT.

CDoT starts computing $\max_{l_V} \omega(l_V)$ on the vertices that are farther from the tree root and memoizes these results. On a leaf V, there are no children to take into consideration and the label that maximizes the reward is simply the first label in $\mathcal{L}_{\sqsubseteq}(V)$. If V is not a leaf, we assume to have at hand some procedure to evaluate $\omega(l_W)$ for each label belonging to a child W of V. Let l_V be the first (more promising) label in $\mathcal{L}_{\sqsubseteq}(V)$ and let us compare it with the next most promising label l'_V. The main insight in CDoT is that it is not always necessary to delve into the inspection of l'_V. In fact, Property (3) implies that it is not necessary to inspect l'_V whenever $\omega(l_V) \geq \widetilde{\omega}\,(l'_V)$. Also, Property (2) implies that if indeed $\omega(l_V) \geq \widetilde{\omega}\,(l'_V)$, then $\omega(l_V)$ is necessarily larger than all remaining labels in $\mathcal{L}_{\sqsubseteq}(V)$. If $\nu(\cdot)$ is a strong predictor of optimal labels, it is thus likely that the inequality holds and much computation can be saved.

We now consider how to efficiently compute $\omega(l_V)$ through a process similar to that used for $\max_{l_V} \omega(l_V)$. In the forthcoming discussion, we say that we *open* l_V the first time we actually compute its reward $\omega(l_V)$; vice versa, a label l_V is said to be closed if it has never been opened. Whenever we open a label, we memoize its value and assume to have $O(1)$ time access to $\omega(l_V)$ in subsequent calls. The computationally expensive part in the definition of $\omega(l_V)$ is the evaluation of

$\sum_{W \in \mathrm{ch}(V)} \max_{l_W} [\tau(l_V \twoheadrightarrow l_W) + \omega(l_W)]$. Importantly, each maximization inside the summation can be dealt with independently of the others. Let $l^\star$ be the first, most promising, label in $\mathcal{L}_{\sqsubseteq}(W)$ and l_W be the next label in $\mathcal{L}_{\sqsubseteq}(W)$. Remarkably, $l^\star$ has to be a previously opened label: it belongs to a vertex already processed (it is farther from the root), and it is the most promising label for that vertex. In contrast, l_W can be either open or closed. If it is open, we can choose between the two labels by comparing $\omega(l_W)+\tau(l_V \twoheadrightarrow l_W)$ with $\omega(l^\star)+\tau(l_V \twoheadrightarrow l^\star)$ in $O(1)$. If it is closed, computing the best of the two labels can be expensive. However, we can avoid opening l_W and rule it out as a candidate anyway if

$$\omega(l^\star) + \tau(l_V \twoheadrightarrow l^\star) \geq\!\!\!\widetilde{\omega}\ (l_W) + \tau(l_V \twoheadrightarrow l_W). \tag{4}$$

If additionally it holds $\omega(l^\star) + \tau(l_V \twoheadrightarrow l^\star) \geq\!\!\!\widetilde{\omega}\ (l_W) + \mathcal{T}$, then *a fortiori* Inequality (4) holds for all subsequent labels in $\mathcal{L}_{\sqsubseteq}(W)$, and $l^\star$ is the best possible choice for W.

CDoT is described by Algorithms 1, 2, and 3. Algorithm 1 is the main procedure implementing CDoT. It traverses the input tree starting from the deepest level up to the root. For each level λ (i.e., the set of vertices at depth λ), the algorithm calls the procedure described in Algorithm 2 to determine the best assignment $l_V^\star$ for each vertex V in that level. The output of the algorithm is the reward of the best label of the root vertex. Standard dynamic programming techniques can be used to track the assignments to children vertices. The optimal assignment for each vertex can be then reconstructed in linear time by executing a visiting algorithm on the tree.

Algorithm 2 (process_vertex) computes $\max_{l_V} \omega(l_V)$: the best assignment for a given vertex V. It iterates through the labels associated to vertex V computing their reward by means of Algorithm 3 and keeping aside the best label found so far. It returns the best label as soon as it finds that no other label can possibly ameliorate its reward, i.e., as soon as $\omega(l^\star) >\!\!\!\widetilde{\omega}\ (l_V)$ (Algorithm 2, line 4).

Algorithm 3 (open) computes $\omega(l_V)$: the reward for label l_V of vertex V. The algorithm iterates over all children of vertex V integrating the contributions of each one into $\omega(l_V)$ which is initially set to $\nu(l_V)$. In analysing each child label l_W, the algorithm stops as soon as it verifies that current and all forthcoming labels cannot contribute more than the best child label $l^\star$ to the final outcome, i.e., it breaks from the loop as soon as the current best reward $\omega^\star$ for this child is larger than $\widetilde{\omega}\ (l_W) + \mathcal{T}$ (Algorithm 3, line 7). When this condition is not met, the algorithm has the chance of saving some work anyway: it can exclude that the current label is optimal in $O(1)$ by checking if $\omega^\star >\!\!\!\widetilde{\omega}\ (l_W) + \tau(l_V \twoheadrightarrow l_W)$ (Algorithm 3, line 8). Only when also this condition is not met, the algorithm recursively opens the child label.

4 Algorithm Complexity

Let us consider the final step of an execution of CDoT, and assume that for each vertex V, exactly k_V labels have been opened. In this discussion we separately

Algorithm 1. (**CDoT**) Given a tree T, it returns the reward of the best assignment to its vertices

input a tree T
1: **for** $\lambda \leftarrow depth(T) \dots 0$ **do**
2: **for each** vertex V at level λ **do**
3: $l_V^\star \leftarrow$ process_vertex(V)
4: **end for**
5: **end for**
6: **return** $\omega\left(l^\star_{\mathrm{root}(T)}\right)$

Algorithm 2. (**process_vertex**) Given a vertex V, it computes $\max_{l_V} \omega(l_V)$ and returns the assignment for V that attains the maximum value

input a vertex V
1: $l^\star \leftarrow$ **dummy label**
2: $\omega^\star \leftarrow -\infty$
3: **for all** l_V in list $\mathcal{L}_{\sqsubseteq}(V)$ **do**
4: **break if** $\omega^\star \geq\!\!\!\sim \omega\,(l_V)$
5: open(l_V)
6: **if** $\omega(l_V) > \omega^\star$ **then**
7: $l^\star \leftarrow l_V$
8: $\omega^\star \leftarrow \omega(l_V)$
9: **end if**
10: **end for**
11: **return** $l^\star$

consider the time spent to process each vertex of the graph. We define the quantity $\mathbb{T}(V)$ to represent the overall number of steps spent by Algorithms 2 and 3 to process vertex V. Let us define:

$a(l_V)$**:** the number of steps needed by Algorithm 2 to process label l_V;
$b(l_V)$**:** the number of steps needed by Algorithm 3 to find the best set of children for label l_V.

We note that $a(l_V)$ does not include the time spent by Algorithm 3 since such time is accounted for by $b(l_V)$. Similarly, $b(l_V)$ does not include neither the time spent by Algorithm 2, nor the time spent by recursive calls to Algorithm 3. In fact, the time spent in recursive calls is taken into account by b values of vertices in previous layers. Then we can compute $\mathbb{T}(V)$ as $\mathbb{T}(V) = \sum_{l_V} a(l_V) + b(l_V)$.

Property 1. $\sum_{l_V} a(l_V)$ is at worst $O(k_V)$.

Proof. Since only k_V labels have been opened at the end of the algorithm, and Algorithm 2 does not do any work on closed labels, the number of steps needed to analyze a vertex V by (the loop in) Algorithm 2 cannot be larger than k_V.

We notice that we are overestimating the cost to analyze each vertex, since k_V is the *overall* number of vertices opened either by Algorithm 2 or by Algorithm 3.

Algorithm 3. (**open**) Given a label l_V, it computes $\omega(l_V)$

input a label l_V
1: $\omega(l_V) \leftarrow \nu(l_V)$
2: **for all** $W \in \text{children}(V)$ **do**
3: $l^\star \leftarrow$ **dummy label**
4: $\omega^\star \leftarrow -\infty$
5: **for all** l_W in list $\mathcal{L}_\sqsubseteq(W)$ **do**
6: **if** l_W is closed **then**
7: **break if** $\omega^\star >\widetilde{\omega}\ (l_W) + \mathcal{T}$
8: **next if** $\omega^\star >\widetilde{\omega}\ (l_W) + \tau(l_V \twoheadrightarrow l_W)$
9: open(l_W)
10: **end if**
11: **if** $\omega^\star < \omega(l_W) + \tau(l_V \twoheadrightarrow l_W)$ **then**
12: $l^\star \leftarrow l_W$
13: $\omega^\star \leftarrow \omega(l_W) + \tau(l_V \twoheadrightarrow l_W)$
14: **end if**
15: **end for**
16: $\omega(l_V) \leftarrow \omega(l_V) + \omega^\star$
17: **end for**

However, this overestimation simplifies the following argument without hindering the result.

Property 2. $\sum_{l_V} b(l_V)$ is $O(k_V \sum_{W \in \text{ch}(V)} k_W)$.

Proof. Since only k_W labels of vertex W have been opened at the end of the algorithm, the two loops in Algorithm 3 iterate altogether at most $\sum_{W \in \text{ch}(V)} k_W$ times. Moreover, since the steps performed by recursive calls are not to be included in $b(l_V)$, all operations are $O(1)$, and the complexity accounted for by $b(l_V)$ is $O(\sum_{W \in \text{ch}(V)} k_W)$. For closed labels $b(l_V)$ is zero since Algorithm 3 would have never been called on such labels. This implies that $\sum_{l_V} b(l_V)$ is at most $O\Big(\sum_{l_V} (I_{[l_V \text{ is open}]} \sum_{W \in \text{ch}(V)} k_W)\Big)$ (where $I_{[x]}$ is 1 if x is true and 0 otherwise), thereby resulting in $\sum_{l_V} b(l_V)$ being at most $O\Big(k_V \sum_{W \in \text{ch}(V)} k_W\Big)$.

Theorem 1. *CDoT has $O(|T|K^2)$ worst case time complexity and $O(|T|K \log K)$ best case time complexity.*

Proof. The complexity of CDoT is $\mathbf{T} = O(|T|K \log K) + \sum_V \mathbb{T}(V)$, where the $O(|T|K \log K)$ term accounts for the time needed to sort the labels in each vertex according to $\sqsubseteq$ and for the time spent by Algorithm 1 to iterate over all the vertices.

By applying Property 1 and Property 2 to the definition of $\mathbb{T}(V)$, we have:

$$\begin{aligned}\mathbb{T}(V) &= \sum_{l_V} a(l_V) + b(l_V) = O(k_V) + O\Big(k_V \sum_{W \in \text{ch}(V)} k_W\Big) \\ &= O\Big(k_V + k_V \sum_{W \in \text{ch}(V)} k_W\Big)\end{aligned}$$

which implies $\mathbf{T} = O(|T|K \log K) + \sum_V O\Big(k_V + k_V \sum_{W \in \mathrm{ch}(V)} k_W\Big)$. By assuming all k_V equal to some constant κ, the above expression reduces to:

$$\begin{aligned}\mathbf{T} &= O(|T|K \log K) + O\Big(\sum_V \kappa + \kappa \sum_{W \in \mathrm{ch}(V)} \kappa\Big) \\ &= O(|T|K \log K) + O\Big(\kappa(1+\kappa) \sum_V \sum_{W \in \mathrm{ch}(V)} 1\Big) \\ &= O(|T|K \log K) + O(\kappa^2(|T|-1))\end{aligned}$$

where the last equality holds since the two summations altogether iterate a number of times equal to the number of edges in the tree. The worst case occurs when CDoT opens every label in every vertex. In such case $\kappa = K$ and the above formula reverts to $O(|T|K^2)$. In the best case CDoT opens only one vertex per layer, $\kappa = 1$ and the complexity is $O(|T|K \log K)$.

5 Related Works

Most of the research in solving the MAP problem deals with the much tougher problem of general graphs. In this context, due to the exponential cost of solving this problem exactly, most attempts focused on approximate techniques that trade accuracy for speed. Among approximate methods, we recall Loopy Belief Propagation [11], Linear Programming Relaxations [12], and Branch and Bound methods [13]. All mentioned approaches obtain better time performance at the price of lower accuracy. Another interesting approach is based on min-cuts of a graph built from the PGM [14]: it allows for exact inference on a certain class of binary variables and degrades to approximate inference otherwise.

Exact methods (e.g., variable elimination [15] and belief propagation [16]) basically work by pushing maximization operations to inner levels of the probability expression so to obtain a rewritten, cheaper to compute, expression. Their computational cost depends both on the time needed for rewriting the probability expression and on the time needed for computing it.

A distinguishing feature of our approach is that we assume that the rewritten expression for calculating the probability is given and we provide an algorithm that further improves its computation. To this regards, we are not aware of any competing approach that works in the case of tree-structured PGMs. In contrast, several algorithms have been proposed for the specific case of sequences (e.g., [9,17,10]).

6 Experiments

Before delving into the details of the experimentation we note that while the main discussion focused on directed graphical models, hereafter, as a way to prove the generality of the approach, we report the experiments on undirected trees.

In order to evaluate the performances of the proposed approach, we compare it with the direct evaluation of Expression (1). To these ends, we implemented a Dynamic Programming algorithm (hereafter referred to as the *DP* algorithm) that evaluates the expression. As mentioned, competing approaches deal with the more general case of unconstrained graphs and it would be unfair to compare them with an algorithm specifically built to work on trees. By implementing a dynamic programming version of Expression (1), we are basically comparing CDoT with the Variable Elimination algorithm, but disregarding the time needed to build the formula.

We performed a number of experiments to assess the time performance of the two algorithms by generating tree structures under a number of different problem settings. The parameters defining each experimental setting are:

$|T|$: the number of vertices in T;
K: the number of labels per vertex;
p_λ: the probability of increasing the tree depth. By setting $p_\lambda = 1$ the graph degenerates into a sequence; by setting it to 0, it degenerates into a graph where all vertices are connected to the root;
$\mu_\nu^\uparrow$, $\mu_\nu^\downarrow$: the set of labels in each vertex has been partitioned into two sets $\nu^\uparrow$ and $\nu^\downarrow$. The weights of the labels in the two sets have been drawn randomly from two Normal distributions. The parameters $\mu_\nu^\uparrow$ and $\mu_\nu^\downarrow$ control the mean values of the two distributions: $\mathcal{N}(\mu_\nu^\uparrow, 100)$ and $\mathcal{N}(\mu_\nu^\downarrow, 25)$;
$|\nu^\uparrow|$: the number of labels sampled from $\mathcal{N}(\mu_\nu^\uparrow, 100)$;
μ_τ, σ_τ: the mean and standard deviation for the weights associated with transitions among labels. Specifically, all transition weights have been sampled from the Normal distribution $\mathcal{N}(\mu_\tau, \sigma_\tau^2)$.

To better illustrate the experimentation we define a base experimental setting and derive the other experimental settings by varying some parameters. The defaults used are as follows.

Parameters:	$\lvert T\rvert$	K	p_λ	$\mu_\nu^\uparrow$	$\mu_\nu^\downarrow$	$\lvert\nu^\uparrow\rvert$	$\lvert\nu^\downarrow\rvert$	μ_τ	σ_τ
Values:	100	100	0.4	100	10	10	$K - \lvert\nu^\uparrow\rvert$	10	5

The setting for the base experiment corresponds to a problem involving 100 vertices (99 edges) and $10,000$ labels, for a total of $990,000$ transitions. By setting $\mu_\nu^\uparrow = 100$, $\mu_\nu^\downarrow = 10$, and $|\nu^\uparrow| = 10$, we define a tree where each vertex has several ($\sim$10%) high-weighted, highly discriminative labels. By setting a $\sigma_\tau = 5$ we define experimental conditions where, on average, the CDoT algorithm is expected to find the optimal solution by inspecting only a fraction of the transitions.

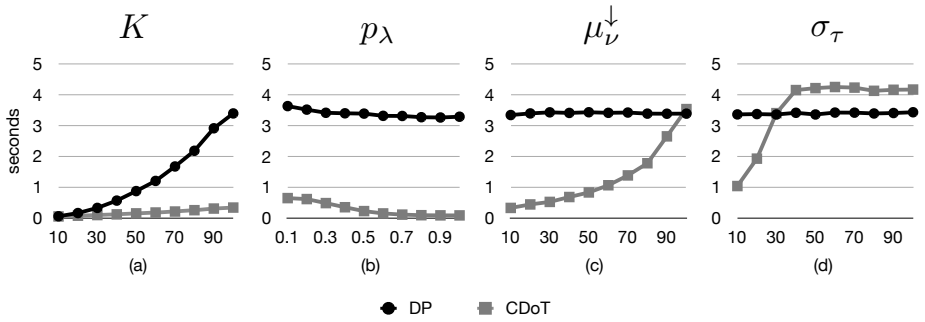
In the experimentation we explore variations to this setting so to compare the performance of the algorithms in different, harder, scenarios.

We generated a total of 100 problem instances for each setting, ran both algorithms on each problem instance and averaged the results.

The algorithms have been implemented in Ruby and run using the standard interpreter (version 1.9.3). All experiments have been executed on computers sporting Intel Xeon dual core CPUs (clock: 2.33GHz, RAM: 8Gb).

7 Results

We compare the performances of CDoT and of the DP algorithm on 36 different settings obtained by varying the following parameters: K, p_λ, $\mu_\nu^\downarrow$, and σ_τ. Results are reported in the following figure:



Panel (a) shows the performances of the two algorithms when varying K in $\{10, 20, \ldots, 100\}$. The figure confirms the complexity analysis reported in Section 4 for the best case scenario: the results show an almost linear dependency of CDoT on K. Vice versa, as expected, the DP algorithm shows a non linear (quadratic) dependency on K.

Panel (b) shows that the computational cost of the DP algorithm does not depend on the branching factor of the tree.[2] Interestingly, CDoT performances slightly improve as the branching factor decreases. To explain this behavior let us focus on the bounds at line 4 of Algorithm 2 and at lines 7 and 8 of Algorithm 3. It can be shown that $\omega(l_V) \geq\!\!\!\!\sim\ \widetilde{\omega}\ (l'_V)$ iff:

$$\nu(l_V) - \nu(l'_V) \geq \sum_{W \in \mathrm{ch}(V)} \mathcal{T} - \tau(l_V \twoheadrightarrow l_W^{\twoheadrightarrow}) + \omega(l_W^\star) - \omega(l_W^{\twoheadrightarrow}) \tag{5}$$

where $l_W^\star$ is the best label assigned to the root of the subtree rooted in W, and $l_W^{\twoheadrightarrow}$ is the best label in W for transitions starting from l_V, i.e.: $l_W^{\twoheadrightarrow} = \arg\max_{l_W} [\tau(l_V \twoheadrightarrow l_W) + \omega(l_W)]$ and $l_W^\star = \arg\max_{l_W} \omega(l_W)$.
Inequality (5) shows that the bounds are progressively more likely to hold as the number of children of V decreases, thus implying better performances of the CDoT algorithm as the average branching factor decreases.

Panels (c) and (d) show how the performances of the two algorithms vary as the assumptions underlying the CDoT optimization strategy progressively weaken. The CDoT computational cost changes depending on how much the

[2] The Figure actually shows a small increase in the performances of the DP algorithm that is not justified by its complexity analysis. Further experiments show that the observed behavior depends on how hashes handling the transitions in the graph are laid out in memory. Importantly, both algorithms share the same data structures, so they are equally affected by this implementation detail.

$\nu(l_V)$ are able to predict the best possible label (i.e., how likely it is that the difference $\nu(l_V) - \nu(l'_V)$ is large), and on how likely high $\tau(l_V \twoheadrightarrow l_W)$ are associated with high $\nu(l_V)$.

Panel (c) shows that CDoT is very resilient to fluctuations of the likelihood of $\nu(l_V) - \nu(l'_V)$. Even when the difference is very small (e.g., for $\mu_\nu^\downarrow = 90$) the algorithm still outperforms DP. The same does not hold for σ_τ: panel (d) shows that when σ_τ grows larger than 30, CDoT looses its edge over DP. This is the only case where we observe higher costs in running CDoT instead of DP. Performances are *lower* instead of the *same* due to the cost of sorting the labels to build $\mathcal{L}_\sqsubseteq(V)$ (Algorithm 2 – line 3, and Algorithm 3 – line 5). The actual point where the two curves cross depends on the interaction between the parameter σ_τ and the difference $\mu_\nu^\uparrow - \mu_\nu^\downarrow$. Again, Inequality (5) helps to explain this behavior. In fact, $\mu_\nu^\uparrow - \mu_\nu^\downarrow$ controls the likelihood that the left hand side is large; term $\mathcal{T} - \tau(l_V \twoheadrightarrow l_W^{\twoheadrightarrow})$ tends to get larger values as the standard deviation of τ grows.

As mentioned in Section 1, past evidence on sequences corroborates the hypothesis that real world problems often feature strong evidence associated with vertices, while evidence on transitions are mainly useful to discriminate ambiguous cases. In commenting the above findings, we would like to recall that CDoT is badly affected by high transition variances when it is frequent that high-weighted labels are linked by low-weighted transitions. In fact, it can be argued that in real world problems transition weights mostly corroborate the predictions formulated by considering vertex evidence, rather than contradicting them.

Summarizing, while the worst case complexity of CDoT is the same of DP, in our experiments it outperforms DP in most cases. Furthermore, even in a scenario where evidence on transition often contradicts the one on the vertices, running CDoT instead of DP only requires few additional resources.

8 Concluding Remarks

In this paper we introduced CDoT, a novel algorithm for answering MAP queries on tree structured PGMs. We discussed its properties, specifically the reasons underlying its behavior; and studied its asymptotic complexity, showing that it has $O(|T|K \log K)$ best case complexity and that it is never asymptotically worse than previous approaches. We also provided an empirical assessment of its performances. Experiments bolster the theoretical analysis, showing that the algorithm performances substantially improve over a dynamic programming based competitor.

The provided experimentation is large (totaling 3,600 runs of each algorithm) albeit limited to synthetic data. We acknowledge the importance of an experimentation assessing to what extent real problems match the assumptions underlying the CDoT algorithm. These efforts may indeed shed light on important facets of CDoT behavior, facets that can be useful for deciding when it is worth adopting it. We defer to future work such experimentation.

References

1. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
2. Heckerman, D., Horvitz, E., Nathwani, B.: Toward normative expert systems: The Pathfinder project. Knowledge Systems Laboratory, Stanford University (1992)
3. Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., Rother, C.: A comparative study of energy minimization methods for markov random fields with smoothness-based priors. IEEE Trans. Pattern Anal. Mach. Intell. 30(6), 1068–1080 (2008)
4. Sun, Y., Deng, H., Han, J.: Probabilistic models for text mining. In: Aggarwal, C.C., Zhai, C. (eds.) Mining Text Data, pp. 259–295. Springer (2012)
5. Johnson, M., Griffiths, T., Goldwater, S.: Bayesian inference for PCFGs via Markov Chain Monte Carlo. In: Human Language Technologies 2007, pp. 139–146 (2007)
6. Csűrös, M., Miklós, I.: Streamlining and large ancestral genomes in archaea inferred with a phylogenetic birth-and-death model. Mol. Bio. Evol. 26(9) (2009)
7. Paiement, J.F., Eck, D., Bengio, S.: A Probabilistic Model for Chord Progressions. In: Proc. of the 6th Int. Conf. on Music Information Retrieval, London (2005)
8. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inf. Theory 13, 260–269 (1967)
9. Esposito, R., Radicioni, D.P.: CarpeDiem: Optimizing the Viterbi Algorithm and Applications to Supervised Sequential Learning. JMLR 10, 1851–1880 (2009)
10. Belanger, D., Passos, A., Riedel, S., McCallum, A.: Speeding up MAP with Column Generation and Block Regularization. In: Proc. of the ICML Workshop on Inferning: Interactions Between Inference and Learning. Omnipress (2012)
11. Murphy, K., Weiss, Y., Jordan, M.: Loopy belief propagation for approximate inference: An empirical study. In: Proc. of the 15th Conf. on Uncertainty in Art. Intell., pp. 467–475 (1999)
12. Wainwright, M., Jaakkola, T., Willsky, A.: Map estimation via agreement on trees: message-passing and linear programming. IEEE Trans. Inf. Theory 51(11), 3697–3717 (2005)
13. Marinescu, R., Kask, K., Dechter, R.: Systematic vs. non-systematic algorithms for solving the mpe task. In: Proc. of the 9th Conf. on Uncertainty in Artificial Intelligence, pp. 394–402. Morgan Kaufmann Publishers Inc. (2002)
14. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Trans. Pattern Anal. Mach. Intell. 23(11), 1222–1239 (2001)
15. Zhang, N., Poole, D.: Exploiting causal independence in bayesian network inference. JAIR 5, 301–328 (1996)
16. Weiss, Y., Freeman, W.: On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. IEEE Trans. Inf. Theory 47(2), 736–744 (2001)
17. Kaji, N., Fujiwara, Y., Yoshinaga, N., Kitsuregawa, M.: Efficient staggered decoding for sequence labeling. In: Proc. of the 48th Meeting of the ACL, pp. 485–494 (2010)