

A Comparison of Time-Space Schemes

Student: Robert Mateescu

Supervisor: Rina Dechter

School of Information and Computer Science
University of California, Irvine, CA 92697-3425
{dechter, mateescu}@ics.uci.edu

Abstract. We investigate two parameterized algorithmic schemes for graphical models that can accommodate trade-offs between time and space: 1) AND/OR Cutset Conditioning (**AOC(i)**) and 2) Variable Elimination with Conditioning (**VEC(i)**). We show that **AOC(i)** is better than the vanilla versions of **VEC(i)**, and use the guiding principles of **AOC(i)** to improve **VEC(i)**. Finally, we show that the improved versions of **VEC(i)** can be simulated by **AOC(i)**, which emphasizes the unifying power of the AND/OR framework.

1 Introduction

In this paper we compare AND/OR search [1] and alternating elimination and conditioning controlled by induced-width w (**VEC**) [2, 3]. By analyzing them using the context minimal AND/OR graph data structure [4], we show that **VEC(i)** can be improved via the AND/OR search principle and by careful caching, to the point that both schemes become identically good. We show that the recently proposed AND/OR cutset conditioning [5] (improving cutset, and w -cutset schemes) can simulate any execution of **VEC**, if the latter is augmented with AND/OR search over the conditioning variables.

The analysis is done in the general context of graphical models, assuming no determinism. This is still useful in the context of constraint networks, providing a comparison of the total space that the algorithm might need to traverse.

2 Preliminaries

Definition 1 (graphical model). A graphical model is a 3-tuple $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where: $\mathbf{X} = \{X_1, \dots, X_n\}$ is a set of variables; $\mathbf{D} = \{D_1, \dots, D_n\}$ is the set of their finite domains of values; $\mathbf{F} = \{f_1, \dots, f_r\}$ is a set of real-valued functions.

Definition 2 (pseudo tree). A pseudo tree of a graph $G = (\mathbf{X}, E)$ is a rooted tree T having the same set of nodes \mathbf{X} , such that every arc in E is a backarc in T (i.e., it connects nodes on the same path from root).

Definition 3 (induced graph and induced width). An ordered graph is a pair (G, d) , where G is an undirected graph, and $d = (X_1, \dots, X_n)$ is an ordering of the nodes. The width of a node in an ordered graph is the number of neighbors that precede it in the ordering. The width of an ordering d , denoted $w(d)$, is the maximum width over

all nodes. The induced width of an ordered graph, $w^*(d)$, is the width of the induced ordered graph obtained as follows: for each node, from last to first in d , its preceding neighbors are connected in a clique. The induced width of a graph, w^* , is the minimal induced width over all orderings. The induced width is equal to the treewidth of a graph.

3 Description of Algorithms

AOC and **VECare** are both parameterized memory intensive algorithms that need to use space in order to achieve the worst case time complexity of $O(n k^{w^*})$, where k bounds domain size. The task that we consider is one that is equivalent to solutions counting.

3.1 AND/OR Cutset Conditioning - AOC

The AND/OR search space is a recently introduced [1, 4, 5] unifying framework for advanced algorithmic schemes for graphical models. Its main virtue consists in exploiting independencies between variables during search, which can provide exponential speedups over traditional search methods oblivious to problem structure.

Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, its primal graph G and a pseudo tree \mathcal{T} of G , the associated AND/OR search tree has alternating levels of OR and AND nodes. The OR nodes are labeled correspond to branching according to values of variables, while the AND nodes correspond to problem decomposition. The structure of the AND/OR search tree is based on the underlying pseudo tree \mathcal{T} . The AND/OR search tree can be traversed by a depth first search algorithm, thus using linear space.

Theorem 1 ([6–8, 5]). *Given a graphical model \mathcal{M} and a pseudo tree \mathcal{T} of depth m , the size of the AND/OR search tree based on \mathcal{T} is $O(n k^m)$, where k bounds the domains of variables. A graphical model of treewidth w^* has a pseudo tree of depth at most $w^* \log n$, therefore it has an AND/OR search tree of size $O(n k^{w^* \log n})$.*

The AND/OR search tree may contain *unifiable* nodes, that root identical conditioned subproblems. When unifiable nodes are merged, the search space becomes a graph. The depth first search algorithm can therefore be modified to cache previously computed results, and retrieve them when the same nodes are encountered again. Some unifiable nodes can be identified based on their *contexts* [8]. We only use caching based on *OR context*, denoted as $context(X) = [X_1 \dots X_k]$, which is the set of ancestors of X in \mathcal{T} ordered descendingly, that are connected in the primal graph to X or to descendants of X . The *context minimal* AND/OR graph is obtained by merging all the context unifiable OR nodes. An example will appear later in Figure 4.

Theorem 2 ([7, 1]). *Given a graphical model \mathcal{M} , its primal graph G and a pseudo tree \mathcal{T} , the size of the context minimal AND/OR search graph based on \mathcal{T} is $O(n k^{w_{\mathcal{T}}^*(G)})$, where $w_{\mathcal{T}}^*(G)$ is the induced width of G over the depth first traversal of \mathcal{T} , and k bounds the domain size.*

AND/OR Cutset Conditioning (**AOC**) [5] is a search algorithm that combines AND/OR search spaces with cutset conditioning. The conditioning (cutset) variables

form a *start* pseudo tree. The remaining variables (not belonging to the cutset), have bounded conditioned context size that can fit in memory.

Given a primal graph G , of a graphical model and a pseudo tree \mathcal{T} of G , a *start pseudo tree* \mathcal{T}_{start} is a connected subgraph of \mathcal{T} that contains the root of \mathcal{T} .

We can now define algorithm **AOC(i)**, that depends on a parameter i that bounds the maximum size of a context that can fit in memory. Given a pseudo tree \mathcal{T} , we first find a start pseudo tree \mathcal{T}_{start} such that the context of any node not in \mathcal{T}_{start} contains at most i variables that are not in \mathcal{T}_{start} . This can be done by starting with the root of \mathcal{T} and then including as many descendants as necessary in the start pseudo tree until the previous condition is met. \mathcal{T}_{start} now forms the cutset, and when its variables are instantiated, the remaining conditioned subproblem has induced width bounded by i . The cutset variables can be explored by linear space (no caching) AND/OR search, and the remaining variables by using full caching, of size bounded by i . The cache tables need to be deleted and reallocated for each new conditioned subproblem.

Adaptive Caching for AND/OR Search The cutset principle inspires a refined caching scheme for AND/OR search, which we will call *adaptive caching* (in the sense that it adapts to the available memory), that caches some values even at nodes with contexts greater than the bound i that defines the memory limit. Lets assume that $context(X) = [X_1 \dots X_k]$ and $k > i$. During search, when variables X_1, \dots, X_{k-i} are instantiated, they can be regarded as part of a cutset. The problem rooted by X_{k-i+1} can be solved in isolation, like a subproblem in the cutset scheme, after the variables X_1, \dots, X_{k-i} are assigned their current values in all the functions. In this subproblem, $context(X) = [X_{k-i+1} \dots X_k]$, so it can be cached within space bounded by i . However, when the search retracts to X_{k-i} or above, the cache table for X needs to be deleted and reallocated when a new subproblem rooted at X_{k-i+1} is solved.

Algorithm **AOC(i)** is essentially an AND/OR search with adaptive caching bounded by i for all variables. The AND/OR search algorithm that caches only the full contexts bounded by i is **AO(i)**.

Proposition 1. *AOC(i) increases space requirements linearly compared to AO(i), but the time savings can be exponential.*

3.2 Variable Elimination with Conditioning - VEC

Variable Elimination with Conditioning (**VEC**) [2, 3] is an algorithm that combines the virtues of both inference and search. **VEC** works by interleaving elimination and conditioning of variables. Typically, given an ordering, it prefers the elimination of a variable whenever possible, and switches to conditioning whenever space limitations require it, and continues in the same manner until all variables have been processed. We say that the conditioning variables form a *conditioning set*, or *cutset* (this can be regarded as a *w-cutset*, where w defines the induced width of the problems that can be handled by elimination). The vanilla version of **VEC** will also be called **VEC-OR** because the cutset is explored by OR search rather than AND/OR. When there are no conditioning variables, **VEC** becomes the well known Variable Elimination (**VE**) algorithm. In this case **AOC** also becomes the usual AND/OR graph search (**AO**).

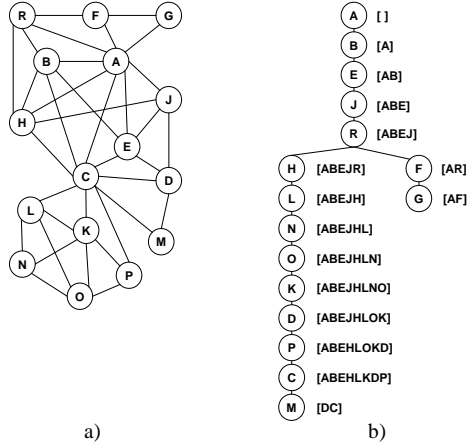


Fig. 1. Primal graph and pseudo tree

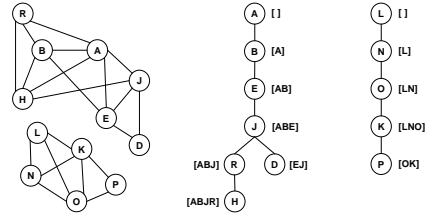


Fig. 2. Components after conditioning on C

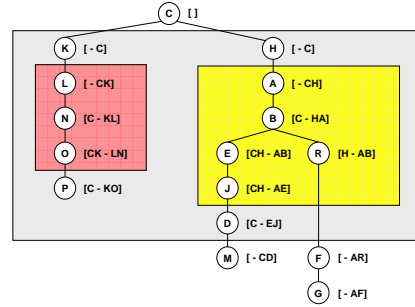


Fig. 3. Pseudo tree for AOC(2)

Theorem 3 (VE and AO are identical [4]). Given a graphical model with no determinism and a pseudo tree, **VE** traverses the full context minimal graph bottom-up by layers (breadth first), while **AO** is a top-down depth-first search that explores (and records) the full context minimal graph as well.

4 AOC(i) Compared to VEC(i)

We will begin by following an example. Consider the graphical model given in Figure 1a having binary variables, the ordering $d_1 = (A, B, E, J, R, H, L, N, O, K, D, P, C, M, F, G)$, and the space limitation $i = 2$. The pseudo tree corresponding to this ordering is given in Figure 1b. The context of each node is shown in square brackets.

If we apply **VEC** along d_1 (processing from last to first), variables G, F and M can be eliminated. However, C cannot be eliminated, because it would produce a function with scope equal to its context, $[ABEHLKDP]$, violating the bound $i = 2$. **VEC** switches to conditioning on C and all the functions that remain to be processed are modified accordingly, by instantiating C . The primal graph has two connected components now, shown in Figure 2. Notice that the pseudo trees are based on this new graph, and their shape change from the original pseudo tree.

Continuing with the ordering, P and D can be eliminated (one variable from each component), but then K cannot be eliminated. After conditioning on K , variables O, N and L can be eliminated (all from the same component), then H is conditioned (from the other component) and the rest of the variables are eliminated. To highlight the conditioning set, we will box its variables when writing the ordering, $d_1 = (A, B, E, J, R, \boxed{H}, L, N, O, \boxed{K}, D, P, \boxed{C}, M, F, G)$.

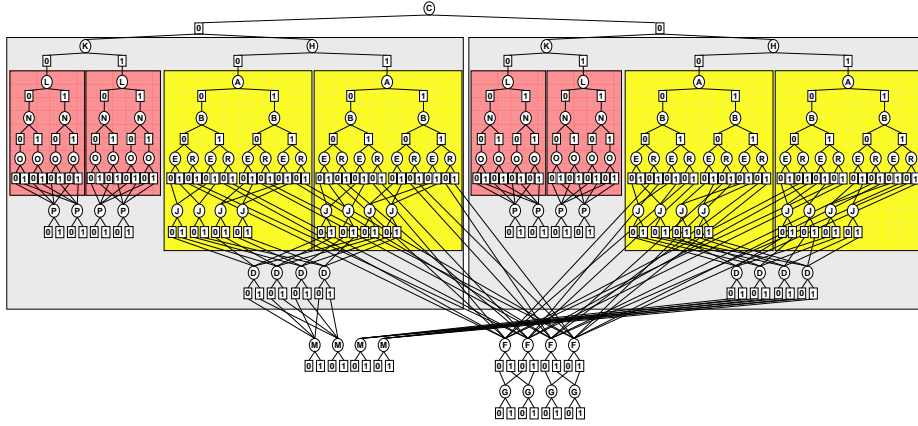


Fig. 4. Context minimal graph

If we take the conditioning set $[HKC]$ in the order imposed on it by d_1 , reverse it and put it at the beginning of the ordering d_1 , then we obtain:

$$d_2 = \left(\underline{[C]}, \underline{[K]}, \underline{[H]}, \underline{[A, B, E, J, R]}_H, \underline{L, N, O}_K, \underline{D, P}_C, M, F, G \right)$$

where the indexed squared brackets together with the underlines represent subproblems that need to be solved multiple times, for each instantiation of the index variable.

So we started with d_1 and bound $i = 2$, then we identified the corresponding conditioning set $[HKC]$ for **VEC**, and from this we arrived at d_2 . We are now going to use d_2 to build the pseudo tree that guides **AOC(2)**, given in Figure 3. The outer box corresponds to the conditioning of C . The inner boxes correspond to conditioning on K and H , respectively. The context of each node is given in square brackets, and the 2-context is on the right side of the dash. For example, $context(J) = [CH-AE]$, and $2-context(J) = [AE]$. The context minimal graph corresponding to the execution of **AOC(2)** is shown in Figure 4.

We can now follow the execution of both **AOC** and **VEC** along this context minimal graph. After conditioning on C , **VEC** solves two conditioned subproblems (one for each value of C), which are the ones shown on gray backgrounds. However, the vanilla version **VEC-OR** is less efficient than **AOC**, because it uses an OR search over the cutset variables, rather than AND/OR. In our example, the subproblem on A, B, E, J, R would be solved eight times by **VEC-OR**, once for each instantiation of C, K and H , rather than four times. It is now easy to make the first improvement to **VEC**, so that it uses an AND/OR search over the conditioning set, an algorithm we call **VEC-AO**.

Let's look at one more condition that needs to be satisfied for the two algorithms to be identical. If we change the ordering to $d_3 = (A, B, E, J, R, \underline{[H]}, L, N, O, \underline{[K]}, D, P, F, G, \underline{[C]}, M)$, (F and G are eliminated after conditioning on C), then the pseudo tree is the same as before, and therefore the context minimal graph for **AOC** is still the one shown in Figure 4. However, **VEC-AO** would require more effort, because the elimination of G and F is performed twice now (once for each instantiation of C), rather than once as was for ordering d_1 . This shortcoming can be eliminated by defining a pseudo tree based version for **VEC**, rather than one based on an ordering. The final

algorithm, **VEC(i)** is given below (where $N_G(X_i)$ is the set of neighbors of X_i in the graph G). Note that the guiding pseudo tree is regenerated after each conditioning.

Algorithm VEC(i)

input : $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; $G = \langle \mathbf{X}, E \rangle$; $d = (X_1, \dots, X_n)$; i

output: Solutions count.

generate the bucket tree \mathcal{T} for d ;

while \mathcal{T} not empty **do**

if $(\exists X_i \text{ leaf in } \mathcal{T}) \wedge (|N_G(X_i)| \leq i)$ **then** eliminate X_i **else** pick X_i leaf of \mathcal{T} ;

for each $x_i \in D_i$ **do**

 assign $X_i = x_i$;

 call **VEC(i)** on each connected component of conditioned subproblem

Theorem 4 (AOC(i) can simulate VEC(i)). *Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ and an execution of **VEC(i)**, there exists a pseudo tree that guides an execution of **AOC(i)** that traverses the same context minimal graph.*

5 Conclusion

We have compared two parameterized algorithmic schemes for graphical models that can accommodate time-space trade-offs. They have emerged from seemingly different principles: **AOC(i)** is search based and **VEC** combines search and inference.

We show that if the graphical models contain no determinism, **AOC(i)** can have a smaller time complexity than the vanilla versions of **VEC(i)**. This is due to a more efficient exploitation of the graphical structure of the problem through AND/OR search, and the adaptive caching scheme that benefits from the cutset principle. These ideas can be used to enhance **VEC(i)**. We show that if **VEC(i)** uses AND/OR search over the conditioning set and is guided by the pseudo tree data structure, then there exists an execution of **AOC(i)** that is identical to it. AND/OR search with adaptive caching (**AOC(i)**) emerges therefore as a unifying scheme, never worse than **VEC(i)**. All the analysis was done by using the context minimal data structure, which provides a powerful methodology for comparing the algorithms.

References

1. Dechter, R., Mateescu, R.: Mixtures of deterministic-probabilistic networks and their and/or search space. In: UAI'04. (2004) 120–129
2. Rish, I., Dechter, R.: Resolution vs. search; two strategies for sat. *Journal of Automated Reasoning* **24(1/2)** (2000) 225–275
3. Larrosa, J., Dechter, R.: Boosting search with variable-elimination. *Constraints* **7(3-4)** (2002) 407–419
4. Mateescu, R., Dechter, R.: The relationship between and/or search and variable elimination. In: UAI'05. (2005) 380–387
5. Mateescu, R., Dechter, R.: And/or cutset conditioning. In: IJCAI'05. (2005) 230–235
6. Freuder, E.C., Quinn, M.J.: Taking advantage of stable sets of variables in constraint satisfaction problems. In: IJCAI'85. (1985) 1076–1078
7. Bayardo, R., Miranker, D.: A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In: AAAI'96. (1996) 298–304
8. Darwiche, A.: Recursive conditioning. *Artificial Intelligence* **125(1-2)** (2001) 5–41