The Impact of AND/OR Search Spaces on Constraint Satisfaction and Counting

Rina Dechter and Robert Mateescu

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

Abstract. The contribution of this paper is in demonstrating the impact of AND/OR search spaces view on solutions counting. In contrast to the traditional (OR) search space view, the AND/OR search space displays independencies present in the graphical model explicitly and may sometimes reduce the search space exponentially. Empirical evaluation focusing on counting demonstrates the spectrum of search and inference within the AND/OR search spaces.

1 Introduction

The primary contribution of this paper is in viewing search for constraint processing in the context of *AND/OR search spaces* rather than *OR spaces*. We demonstrate how the AND/OR principle can exploit independencies in the graph model to yield exponentially smaller search spaces. The notion of AND/OR search tree is closely related to the notion of pseudo-tree rearrangement introduced in [1] for constraint satisfaction. In recent work we revive this idea, extend it to various tasks for any graphical model and extend AND/OR spaces to search graphs as well, thus allowing caching. In this paper we focus on counting for constraint networks and provide initial empirical evaluation along the full spectrum of space and time.

2 AND/OR Search Trees

In the following sections we will use the common definitions and notations for constraint networks and their associated parameters. For more details see [2].

Definition 1 (AND/OR search tree based on a DFS tree). *Consider a constraint network $\mathcal{R}$ and a DFS spanning tree T of its primal graph. The AND/OR search tree of $\mathcal{R}$ based on T, denoted S_T, has alternating levels of OR nodes (labeled with variable names, e.g. X) and AND nodes (labeled with variable values, e.g. $\langle X, v\rangle$). The root of S_T is an OR node labeled with the root of T. The children of an OR node X are AND nodes, each labeled with a value of X, $\langle X, v\rangle$. The children of an AND node $\langle X, v\rangle$ are OR nodes, labeled with the variables that are children of X in T.*

Consider the tree T in Fig. 1 describing a graph coloring problem over domains $\{1, 2, 3\}$. Its traditional OR search tree along the DFS ordering $d = (X, Y, T, R, Z, L, M)$ is given in Fig. 2, its AND/OR search tree based on the DFS tree T and a highlighted solution subtree are given in Fig. 3.

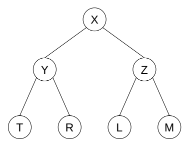

Fig. 1. Tree T

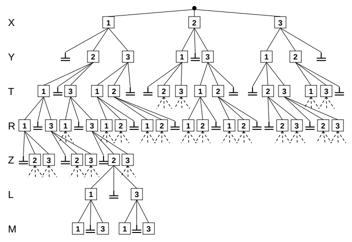

Fig. 2. OR search tree

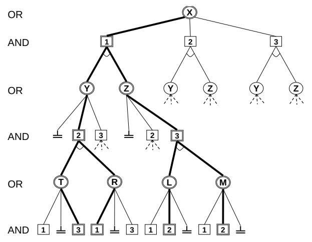

Fig. 3. AND/OR search tree

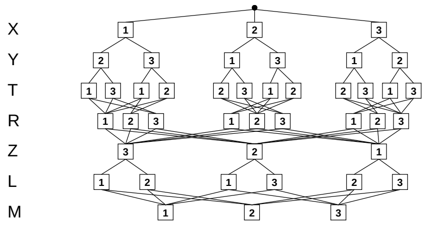

Fig. 4. Minimal OR search graph of the tree problem in Fig. 1

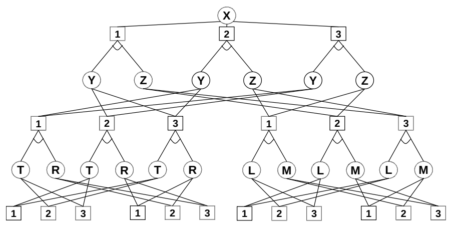

Fig. 5. Minimal AND/OR search graph of the tree problem in Fig. 1

Pseudo-trees. The construction of AND/OR search trees can use as its basis not just DFS spanning trees, but also the more general *pseudo-trees* [1, 3]. They have the property that every arc of the original primal graph is a back-arc in the pseudo-tree (i.e. it doesn't connect across different branches). Clearly, any DFS tree and any chain are pseudo-trees. Searching the OR space corresponds to searching a chain. It is easy to see that searching an AND/OR tree is exponential in the depth of the pseudo-tree. Also, it is known that the minimal depth over pseudo-trees, m^*, satisfies $m^* \leq w^* \log n$ [3].

Theorem 1. *Given a constraint network $\mathcal{R}$ and a pseudo-tree T, its AND/OR search tree S_T is sound and complete (contains all and only solutions) and its size is $O(n \cdot \exp(m))$ where m is the depth of the pseudo-tree. A constraint network that has a tree-width w^* has an AND/OR search tree whose size is $O(\exp(w^* \cdot \log n))$.*

3 AND/OR Search Graphs

It is often the case that certain states in the search tree can be merged because the subtrees they root are identical. Any two such nodes are called *unifiable*, and when merged, transform the search tree into a search graph. It can be shown that the closure of an AND/OR search graph under the merging of unifiable states yields a unique fixed point, called the *minimal AND/OR search graph*. Merging is applicable to the OR search space as well. However, in many cases it will not be able to reach the compression we see in the AND/OR search graph. Fig. 4 and Fig. 5 show a comparison between minimal OR and AND/OR search graphs for the problem in Fig. 1. Note that in the AND/OR graph only the AND levels are relevant, the OR levels serving only for clarity.

We will now describe some efficient rules for generating AND/OR search graphs. The idea is to extract from each path only the relevant *context* that completely deter-

```
procedure AND-OR-COUNTING
Input: A constraint network; a pseudo-tree $T$ of its constraint graph; parents $pa_i$ and parent-separators $psa_i$.
Output: The number of solutions $g(X_0)$. $\pi$ denotes the current partial assignment path.
1.    Initialize: $X_0 = root(T), type(X_0) = OR$, OPEN $\leftarrow X_0, cache \leftarrow \phi$;
2.    Expand:   $n \leftarrow$ first node in OPEN; generate all successors of $n$ as follows:
        if ($type(n) == OR$), denote $n = X$
          succ(X) $\leftarrow \{\langle X, v\rangle \mid consistent(\langle X, v\rangle)\}$
          if ($succ(X) = \phi$) then $g(X) = 0$; (dead-end)
            [ $cache(\pi_{pa_X}) = 0$ , update constraints and go to step 3 ]
          for each $\langle X, v\rangle \in succ(X)$ do $\pi' \leftarrow \pi \cup (\langle X, v\rangle)$
            [ if ($cache(\pi'_{psa_X}) \neq \phi$) then $g(\langle X, v\rangle) = cache(\pi'_{psa_X})$ else ]  add $\langle X, v\rangle$ to OPEN
        if ($type(n) == AND$), denote $n = \langle X, v\rangle$
          if $X$ is a leaf in $T$ then, $g(\langle X, v\rangle) = 1$, go to step 3
          $succ(\langle X, v\rangle) \leftarrow \{Y \mid Y \in children(X)\ in\ T\}$
          for each $Y \in succ(\langle X, v\rangle)$ do
            [ if ($cache(\pi_{pa_Y}) \neq \phi$) then $g(Y) = cache(\pi_{pa_Y})$ else ]  add $Y$ to OPEN
3.    Propagate:   while you can propagate $g$ values:
        a. For a non-terminal AND node $\langle X, v\rangle$:
          if ($Y \in succ(\langle X, v\rangle)$ and $g(Y) = 0$), remove siblings of $Y$ from OPEN, $g(\langle X, v\rangle) = 0$.
          if all $succ(\langle X, v\rangle)$ are evaluated, $g(\langle X, v\rangle) = \Pi_{Y \in succ(\langle X, v\rangle)} g(Y)$
          [ if ($\langle X, v\rangle$ is evaluated) then $cache(\pi_{psa_X}) = g(\langle X, v\rangle)$ ]
        b. For a non-terminal OR node $X$:
          if all succ(X) have $g$ values, $g(X) = \Sigma_{\langle X, v\rangle \in succ(X)} g(\langle X, v\rangle)$
OR           [ if ($X$ is evaluated) then $cache(\pi_{pa_X}) = g(X)$ ]
4.    if $X_0$ was evaluated, exit with $g(X_0)$ else go to step 2.
```

Fig. 6. The counting algorithm

mines the unexplored portion of the space. Subsequently, if memory allows, the subgraph is only solved once and the results are indexed by the context and cached. We will need some more definitions.

Definition 2 (induced-width relative to a pseudo-tree). *Given G^T, which is an extended graph of G that includes all the arcs in the pseudo-tree T, the* induced width of G relative to the pseudo-tree T, *$w_T(G)$, is the induced-width of G^T along the DFS ordering of T.*

Definition 3 (parents, parent-separators). *Given the induced-graph, ${G^*}^T$ of an extended graph G^T, the parents of X denoted pa_X, are its earlier neighbors in the induced-graph. Its parent-separators, psa_X are its parents that are also neighbors of future variables in T.*

In ${G^*}^T$, the parent-separators of every node X_i separate in T its ancestors on the path from the root, and all its descendents in G^T. Therefore, any two nodes having the same context, that is, the same assignments to their parent-separators, can be merged.

Theorem 2. *Given G, a pseudo-tree T and its induced width $w = w_T(G)$, the size of the AND/OR search graph based on T obtained when every two nodes in S_T having the same context are merged is $O(n \cdot k^w)$, when k bounds the domain size.*

Thus, the minimal AND/OR search graph of G relative to T is $O(n \cdot k^w)$ where $w = w_T(G)$. Since, as can be shown, $\min_T\{w_T(G)\}$ equals the tree-width w^* and since $\min_{T \in chain}\{w_T(G)\}$ equals the path-width pw^* we obtain that the minimal AND/OR search graph is bounded exponentially by the primal graph's tree-width, while the minimal OR search graph is bounded exponentially by its path-width. It is well known [4] that for any graph $w^* \leq pw^* \leq w^* \cdot \log n$. It is also easy to place m^* (the minimal pseudo-tree depth) yielding $w^* \leq pw^* \leq m^* \leq w^* \cdot \log n$.

Table 1. A/O FC, N=60, K=3

N=40, K=3, C=50, S=3, 20 inst., w*=13, d=20							
		Time			Number of dead-ends		
tightness		20%	40%	60%	20%	40%	60%
# solutions		0	0	147898575	0	0	147898575
	BE	8.714	8.709	**8.637**			
i=0	A/O FC	**0.030**	**0.454**	32.931	533	9,229	1,711,947
	OR FC	0.031	0.511	9737.823	533	9,897	324,545,908
i=6	A/O FC	0.029	0.454	25.140	533	8,991	917,612
	OR FC	0.032	0.508	7293.472	533	9,897	208,159,068
i=13	A/O FC	0.030	0.457	**11.974**	533	8,533	**181,157**
	OR FC	0.032	0.494	1170.203	533	9,283	**20,018,823**

Table 2. A/O FC, N=100, K=2

N=100, K=2, C=130, S=3, 20 inst., w*=32, d=43				
tightness	10%	30%	50%	70%
# solutions	0	0	0	0
Time (seconds)				
i=20	0.069	0.193	3.572	677.045
Number of nodes				
i=20	70	406	4,264	1,139,860
Number of dead-ends				
i=20	72	204	4,266	1,043,692

4 AND/OR Algorithms for Counting

Figure 6 presents the basic DFS traversal of the AND/OR search space. The square bracketed lines allow different levels of caching. The nodes in the search graph are labeled by g-values. These stand for the number of solutions below that variable (or variable-value). The computation of the number of solutions is done at step 3 by multiplication (for AND nodes) and summation (for OR nodes). The complexity is, [2]:

Theorem 3. AND-OR-COUNTING *with linear space has time complexity* $O(n \cdot \exp(w^* \cdot \log n))$*, where* w^* *is the tree-width of the problem. With full caching, it is time and space exponential in* w^**. For OR space, the complexity is exponential in the path-width.*

5 Empirical Demonstration

We ran a version of the counting algorithm, which uses forward checking (FC) as the constraint propagation method, defined by the *consistent* function in step 2 of the algorithm. We compared AND/OR and OR search spaces, resulting in two algorithms: A/O FC and OR FC. We tried different levels of caching, controlled by an *i-bound* which defines the maximum context size that can be cached. We also compared against bucket elimination (BE) in some cases, where space was available. We report average measures over 20 instances. Also, $w*$ is the induced width and d is the depth of the pseudo-tree. The constraint networks were generated randomly uniformly given a number of input parameters: N - number of variables; K - number of values per variable; C - number of constraints; S - the scope size of the constraints; t - the tightness (percentage of allowed tuples per constraint). For extended results see [2].

Table 1 shows a comparison on moderate size problems which allowed bucket elimination to run. The bolded time numbers show the best values in each column. The most important thing to note is the vast superiority of AND/OR space over the traditional OR space. A/O FC and OR FC are comparable only on inconsistent problems (up to $t = 40\%$). When the problems are consistent, the difference becomes greater with increasing number of solutions. For BE we only report time, which is not sensitive to the tightness of the problems.

Table 2 shows examples of large networks for which BE and traditional OR search were infeasible. We ran only A/O FC with the maximum cache size possible for our machine. This shows that AND/OR search is more flexible, being able to solve problems of much larger size than inference algorithms or the traditional OR search.

6 Conclusions, Discussion and Related Work

The paper shows how counting algorithms can be affected when formulated as searching AND/OR search trees and graphs rather than searching their OR counterparts. We present and analyze counting algorithms and provide initial empirical evaluation along the full spectrum of space and time. We compare counting algorithms on the AND/OR search space when pruning is accomplished by forward-checking and show how their performance is affected by different levels of caching and how it is compared to bucket-elimination, as a function of problem tightness. The empirical evaluation shows that AND/OR search space is always better than the traditional OR space, often yielding exponential improvements. Compared to inference based algorithms (bucket elimination), AND/OR search is more flexible and able to adapt to the amount of available space. All the existing constraint propagation techniques are readily available for AND/OR search. Coupling this with the possibility of caching makes AND/OR search a very powerful scheme. For full details see [2].

Related work. It can be shown that graph-based backjumping [5, 6] mimics the exploration of an AND/OR search tree. Indeed, it was shown that the depth of a DFS-tree or a pseudo-tree [7, 3] plays an important role in bounding backjumping complexity. Memory-intensive algorithms can be viewed as searching the AND/OR search graph, such as recent work [8] which performs search guided by a tree-decomposition for constraint satisfaction and optimization. A similar approach was introduced recently in [9, 10] both for belief updating and counting models of a CNF formula. Relationship between minimal AND/OR graphs and tree-OBDDs can be shown.

Acknowledgments

This work was supported in part by the NSF grant IIS-0086529 and the MURI ONR award N00014-00-1-0617.

References

1. Freuder, E.C., Quinn, M.J.: Taking advantage of stable sets of variables in constraint satisfaction problems. In: International Joint Conference on Artificial Intelligene. (1985) 1076–1078
2. Dechter, R., Mateescu, R.: The impact of AND/OR search spaces on constraint satisfaction and counting. Technical report, UCI (2004)
3. Bayardo, R., Miranker, D.: A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In: AAAI'96. (1996) 298–304
4. Bodlaender, H., Gilbert, J.R.: Approximating treewidth, pathwidth and minimum elimination tree-height. Technical Report RUU-CS-91-1, Utrecht University (1991)
5. Dechter, R.: Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. Artificial Intelligence **41** (1990) 273–312
6. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
7. Freuder, E.C., Quinn, M.J.: The use of lineal spanning trees to represent constraint satisfaction problems. Technical Report 87-41, University of New Hampshire, Durham (1987)
8. Terrioux, C., Jegou, P.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artificial Intelligence **146** (2003) 43–75
9. Darwiche, A.: Recursive conditioning. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence. (1999)
10. F. Bacchus, S.D., Piassi, T.: Value elimination: Bayesian inference via backtracking search. In: Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence. (2003)