Caching in Context-Minimal OR Spaces

Rina Dechter
Dept. of Computer Science
Univ. of California, Irvine
dechter@ics.uci.edu

Levi H. S. Lelis
Departamento de Informática
Universidade Federal de Viçosa, Brazil
levi.lelis@ufv.br

Lars Otten
Univ. of California, Irvine
(now at Google Inc.)
lotten@uci.edu

Abstract

In empirical studies we observed that caching can have very little impact in reducing the search effort in Branch and Bound search over context-minimal OR spaces. For example, in one of the problem domains used in our experiments we reduce only by 1% the number of nodes expanded when using caching in context-minimal OR spaces. By contrast, we reduce by 74% the number of nodes expanded when using caching in context-minimal AND/OR spaces on the same instances. In this work we document this unexpected empirical finding and provide explanations for the phenomenon.

Introduction

Some of the most successful solvers for finding optimal solutions in graphical models (e.g., MPE/MAP, weighted CSPs), explore the context-minimal AND/OR search graph (defined below) using a Depth-first Branch and Bound (DF-BnB) algorithm guided by mini-bucket heuristics (Marinescu and Dechter 2009; Ihler et al. 2012). A commonly used enhancement in this context is caching. Namely, when a subproblem associated with a node is solved, the value of its solution is cached so that, when the subproblem is reached again from a different path, the solution is retrieved from cache and the node is not explored redundantly. We call this a cache hit. Avoiding redundant node exploration and exploring the (context-minimal) AND/OR search *graph*, in contrast to exploring the AND/OR search *tree*, was shown to lead to substantial search speed up because the graph can be much smaller than the tree, at the expense of some memory (Marinescu and Dechter 2009).

Theory suggests significant reduction in search effort in both AND/OR and OR search spaces. In particular, the size of the OR search tree is exponential on the problem's number of variables n, while the context-minimal search graph is exponential in its path-width pw, only. It is known that pw can be far smaller than n (Bodlaender 2007).

Our Contributions

Despite the theoretical expected reduction in search effort due to searching OR graphs, compared with the tree, we recently observed that when searching the OR context-minimal graph using Depth-first Branch and Bound search, that the number of cache hits was minimal, with almost no difference between searching the OR tree and the OR graph. We therefore conducted a systematic empirical study to investigate this observed phenomenon.

Our results on three benchmarks show that effective caching (in terms of cache hits) is indeed almost non-existent in OR graph search and the impact on the search space is minimal. In contrast, when searching the AND/OR space the reduction is highly significant: first, a substantial reduction in size from OR trees to AND/OR trees; second, a further notable reduction moving from AND/OR trees to AND/OR graphs. Both of these reductions are in line with the theory. In particular, our experiments show that the difference in performance between OR and AND/OR searches is more pronounced than it seems to be implied by theory.

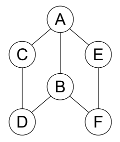
Background

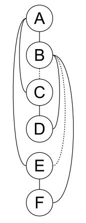
A *graphical model* consists of a set of variables X, their finite domains D, and a set of non-negative real-valued cost functions F defined on subsets of the variables, called function scopes. The function scopes imply a *primal graph* over X where variables that appear in the same function scope are connected. The variables in the primal graph can be ordered, yielding the *induced graph*, where each node's earlier neighbors are connected, with a certain *induced width* w. Cf. (Dechter 2013) for details.

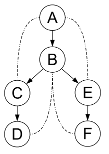
A *pseudo tree* of a graphical model captures problem decomposition and guides the **AND/OR search tree** which consists of alternating levels of OR and AND nodes: OR nodes correspond to variables and AND nodes to value assignments of the OR parent's variable, also rooting conditionally independent subproblems. Edges are annotated by values derived from the input functions F.

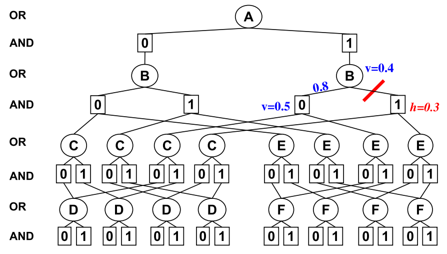
Certain nodes in the search tree root identical subproblems and can be merged, based on their *context*, yielding an **AND/OR search graph**. Intuitively, the context of a variable is the subset of its ancestors that completely separates the subproblem below from the rest of the graph. Thus assigning values to these variables uniquely determines the search space below.

DEFINITION **1 (Context)** *Given a primal graph G and a*


(a) Primal graph.


(b) Induced graph.


(c) Pseudotree.



(d) Context-minimal AND/OR search graph with AOBB pruning example.

Figure 1: Example problem with six variables, induced graph along ordering A, B, C, D, E, F, corresponding pseudotree, and resulting AND/OR search graph with AOBB pruning example.

pseudo tree $\mathcal{T}$ of a graphical model, the context *of a variable X_i are the ancestors of X_i that are connected in G to X_i or to descendants of X_i in $\mathcal{T}$. An assignment of values to X_i and its context variables is called a context instantiation. A* maximal context *is one that's not strictly included in another context.*

DEFINITION **2 (Induced width, path width)** *Given a primal graph G and a pseudo tree $\mathcal{T}$ of a graphical model, the* induced width *of G relative to $\mathcal{T}$ is the maximum width of its induced* pseudo tree *obtained by recursively connecting the parents of each node, going from leaves to root along each branch. In that process we consider both the tree arcs and the arcs in the graphical model. If the pseudo-tree is a chain its induced width is also called* path width.

Example 1 *Figure 2 shows the variable contexts along a chain pseudo tree and corresponding OR graph. For instance, the context of F is B, D, E. A and C do not appear in the context of F because they are not connected to F or the subproblem below.*

AND/OR Branch and Bound (AOBB) (Marinescu and Dechter 2009): AOBB traverses the context-minimal AND/OR graph in a depth-first manner while keeping track of the current lower bound on the optimal solution cost. A node n is pruned if this lower bound exceeds a heuristic upper bound on the solution to the subproblem below n. Consider the example in Fig. 1d where the current lower bound on the best solution at B is $0.8 \cdot 0.5 = 0.4$, and since the upper bound at $B = 1$ is $h = 0.3$ we prune B=1. In this work we employ the admissible and consistent *mini-bucket heuristic*, parametrized by the i-bound that trades of accuracy and complexity (Kask and Dechter 2001; Dechter and Rish 2003).

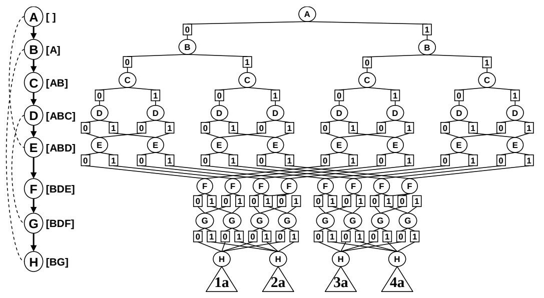


Figure 2: Context minimal graph (full caching)

Dead- and Active-Caches

In a context-minimal AND/OR or OR search space (see Figure 1 (d) and Figure 2) many of the nodes have only a single path from the root to the node. These nodes should not be cached since they will be reached at most once during search; they are called *dead-caches*. For example $(A = 1, B = 1, C = 1)$ in Figure 2 is a dead-cache. The set of dead-caches can be determined using the following rule.

PROPOSITION **1** *(Darwiche 2001) If X is the parent of Y in the pseudo-tree $\mathcal{T}$ and if $context(X) \subset context(Y)$ then $context(Y)$ is a* dead-cache *for any assignment to the context variables. A variable is an* active-cache *if it is not a dead-cache relative to $\mathcal{T}$.*

This can be translated to the following rule: *when a variable has a maximal context relative to a branch in the pseudo tree (a path from root to a leaf), its child variable on that branch is an active cache.* Note that a context can be maximal relative to a branch but not globally maximal.

Example 2 *Inspecting the context of each variable in Figure 2 left, we see, moving from root to leaves, that ABC is a maximal context (there is a single branch here). Thus E is an active cache variable. In this example variables H, G, F, E correspond to active caches while the rest are dead caches.*

Any variable, dead-cache or not, contribute to the context-minimal graph, a number of nodes exponential in its context size. If a variable is an active-cache, its corresponding nodes must be cached during search.

PROPOSITION **2** *The number of nodes in the context-minimal AND/OR search graph along pseudo-tree $\mathcal{T}$, denoted $N(\mathcal{T})$, and the number of active cache nodes denoted, $C(\mathcal{T})$, obey, $N(\mathcal{T}) = \sum_{X_i \in X} k^{|context(X_i)|+1}$ $C(\mathcal{T}) = \sum_{active(X_i) \in X} k^{|context(X_i)|+1}$ We get the ratio $R(\mathcal{T}) = \frac{C(\mathcal{T})}{N(\mathcal{T})}$.*

Computing $N(\mathcal{T})$ and $C(\mathcal{T})$ is easy once the pseudo-tree is determined. The induced-width (see Definition 2) is also very relevant here because the maximum context size is bounded by the induced-width, w of the graph along a pseudo-tree $\mathcal{T}$ (Definitions 1 and 2) we get:

THEOREM **1** *(Dechter and Mateescu 2007) $N(\mathcal{T}) = O(n \cdot k^{w+1})$.*

We can show that the set of maximal contexts, yield a tree-decomposition, called a join-tree decomposition (Dechter 2013). This allows characterizing the active caches in terms of this context-based *join-tree* (for lack of space we omit the join-tree definition).

PROPOSITION **3 (the context-based join-tree)**
Given a pseudo-tree $\mathcal{T}$ of G, its set of (branch-based) maximal contexts, each combined with its variable, form clusters, where each cluster is connected to a parent cluster along the pseudo-tree, yielding the context-based join-tree.

Example 3 *In Figure 2 left The maximal clusters are* $C_H = \{B, H, G\}$, $C_G = \{G, B, D, F\}$, $C_F = \{F, B, D, E\}$, $C_E = \{E, A, B, D\}$, $C_D = \{D, A, B, C\}$, *all connected in a chain.*

PROPOSITION **4 (number of active cache variables)**
Given a pseudo-tree $\mathcal{T}$ of G with its associated context-based join-tree, T, the number of active cache variables along $\mathcal{T}$, equals the number of edges in T.

Corollary 1 *A pseudo-tree has an induced-width w, it must have at least w dead-cache variables.*

Number of Active Caches of OR vs. AND/OR

We can show that moving from a pseudo-tree to an associated pseudo-chain, the context of variables may only increase.

PROPOSITION **5** *If $\mathcal{T}$ is a pseudo-tree with tree-width w and if $\mathcal{L}$ is a pseudo-chain created by depth-first search of $\mathcal{T}$, having path-width pw, then: (1) The context of every variable in $\mathcal{L}$, contains or equals its context in $\mathcal{T}$. Consequently; (2) The size of each context may only increase, implying $pw \geq w$; (3) The number of active cache variables in $\mathcal{L}$ is smaller or equal to that in $\mathcal{T}$; (4) Since $pw \geq w$ we are guaranteed having pw dead-caches in $\mathcal{L}$ which is more than the guaranteed dead-caches in $\mathcal{T}$.*

Example 4 *In Figure 1 the context of D in the AND/OR tree is AB and in the chain it is ACB. The number of active caches stays the same however. Those are: D and E for the pseudo-tree in Figure 1c and are E and F in the chain.*

Ineffective-Caching Phenomenon

As implied by theory and as illustrated in Figure 2, Caching can play an important role in OR search space by reducing its size from exponential in n to exponential in pw. Yet, recently we observed an empirical phenomenon that puzzled us. We observed that, on a collection of problem instances from different benchmarks, when DFBnB with the mini-bucket heuristic traverses the context-minimal OR search graph, there are almost no cache hits (i.e., caching does not reduce the search effort), while caching is quite effective in the case of AND/OR search on the same problem instances. We consider the following two hypotheses to explain this phenomenon.

H1 Caching is ineffective in OR spaces because the context-minimal OR spaces have a small number of active caches.

H2 Caching is ineffective in OR spaces because the search algorithm prunes the search space considerably and visits a subspace which is mostly a tree and not a graph.

Empirical Evaluation

We consider three problem domains: computing haplotypes in genetic analysis (pedigree), protein side-chain prediction (pdb), and randomly generated grid networks.

Table 1 contains detailed results. For each instance we run DFBnB on the context-minimal OR space without ("OR") and with caching ("OR+C") and on the context-minimal AND/OR space without ("AND/OR") and with caching ("AND/OR+C"). For each run we report the following data: the i-bound, maximum domain size k, induced width w, pseudo tree height h, number of variables n, number of nodes cached during search ("Cached"), number of nodes expanded ("Expanded"), running time in seconds ("Time"), ratio between the number of nodes cached and the number of nodes expanded ("Ratio"), number of nodes pruned by the heuristic ("Pruned"), and the number of times a node was retrieved from cache and not expanded ("Hits"). Note that the induced-width w is a tree-width when we talk about AND/OR and path-width when we talk about Or space.

$R(\mathcal{T})$ is the proportion of active-cache nodes in the search space and it does not account for the pruning DFBnB does. By contrast, the Ratio shown in our table accounts for pruning, as it shows the proportion of active-cache nodes encountered during search with respect to the number of nodes expanded by DFBnB.

Analysis and Discussion

On average caching in OR spaces reduces less than 0.1%, 7%, and 1% the number of nodes expanded on instances of grids, pdb, and pedigree, respectively. By contrast, on average, caching in AND/OR spaces reduces 74%, 62%, and 74% the number of nodes expanded on instances of grids, pdb, and pedigree, respectively.

Pedigrees. We observe here that caching has hardly any impact when searching the OR space. In the reported 4 instances the path-widths are very large (112, 90, 232, 294) implying at least this number of dead caches. Also, the underlying context-minimal search graph must be extremely large containing at least $k^{112}, k^{90}, k^{232}, k^{294}$ nodes. This suggests a very small proportion of active-caches. This fact coupled with the observation that only a tiny fraction of the search space was explored by DFBnB, implies that almost all the active caches were pruned. For AND/OR search we observe a much more significant impact of caching. Yet in this case it does not translate to a significant time difference.

Grids Similarly to the pedigrees we observe on grids almost no change in the OR search space with or without caching. Here the path-widths (103, 115, 142, 172) are far larger then the tree-widths in AND/OR space (20, 20, 21, 22, 25) which can explain the observed behavior. Again we observe that the AND/OR search tree is far smaller (explained by the bounded h) and more interestingly, the impact of caching is significant on top of it, reducing the height into

Instance	Space	i	k	w	h	n	Cached	Expanded	Time	Ratio	Pruned	Hits
Pedigree												
pedigree1	OR	13	4	112	297	298	0	5,762	2	0.00000	5,885	0
	OR+C	13	4	112	297	298	1,072	5,534	2	0.19371	5,786	8
	AND/OR	13	4	15	44	298	0	2,480	2	0.00000	2,337	0
	AND/OR+C	13	4	15	44	298	515	990	2	0.52020	1,059	143
pedigree23	OR	13	5	90	308	309	0	191,831	12	0.00000	353,680	0
	OR+C	13	5	90	308	309	356	191,815	12	0.00186	353,672	2
	AND/OR	13	5	25	52	309	0	330,834	12	0.00000	343,667	0
	AND/OR+C	13	5	25	52	309	6,343	98,579	11	0.06434	124,715	29,758
pedigree37	OR	13	5	232	725	726	0	414,737	37	0.00000	570,523	0
	OR+C	13	5	232	725	726	802	414,491	37	0.00193	570,340	2
	AND/OR	13	5	21	56	726	0	770,701	37	0.00000	894,659	0
	AND/OR+C	13	5	21	56	726	11,584	184,039	33	0.06294	466,898	33,214
pedigree39	OR	13	5	294	952	953	0	1,149,622,319	21,137	0.00000	1,148,395,337	0
	OR+C	13	5	294	952	953	2,718	1,149,619,259	21,167	0.00000	1,148,392,739	66
	AND/OR	13	5	22	74	953	0	32,041,954	261	0.00000	31,313,259	0
	AND/OR+C	13	5	22	74	953	12,104	2,779,832	29	0.00435	3,022,123	593,721
Grid												
50-15-3	OR	10	2	115	224	225	0	11,688,665	161	0.00000	11,342,341	0
	OR+C	10	2	115	224	225	638	11,687,717	161	0.00005	11,341,998	15
	AND/OR	10	2	21	63	225	0	586,344	5	0.00000	478,202	0
	AND/OR+C	10	2	21	63	225	3,661	291,174	4	0.01257	306,756	55,333
50-17-5	OR	10	2	172	288	289	0	437,771,461	7,095	0.00000	407,678,359	0
	OR+C	10	2	172	288	289	1,599	437,769,944	7,153	0.00000	407,677,607	26
	AND/OR	10	2	25	70	289	0	21,947,490	180	0.00000	17,290,691	0
	AND/OR+C	10	2	25	70	289	7,664	6,027,678	55	0.00127	5,507,515	1,288,041
75-16-5	OR	10	2	142	255	256	0	695,049,287	10,778	0.00000	578,543,000	0
	OR+C	10	2	142	255	256	1,109	695,048,407	10,803	0.00000	578,542,766	18
	AND/OR	10	2	22	68	256	0	7,932,925	63	0.00000	4,994,048	0
	AND/OR+C	10	2	22	68	256	11,633	1,021,966	10	0.01138	777,653	309,421
75-16-7	OR	10	2	142	255	256	0	1,317,068,856	20,013	0.00000	862,806,093	0
	OR+C	10	2	142	255	256	896	1,317,067,957	20,029	0.00000	862,805,822	16
	AND/OR	10	2	22	68	256	0	6,977,687	56	0.00000	5,021,139	0
	AND/OR+C	10	2	22	68	256	5,239	1,944,310	18	0.00269	1,969,161	338,314
Pdb												
pdb1aly	OR	3	81	39	121	122	0	549,850,965	21,580	0.00000	5,894,820,705	0
	OR+C	3	81	39	121	122	86,005,397	547,195,149	22,492	0.15717	5,852,566,335	1,327,392
	AND/OR	3	81	11	25	122	0	84,022	11	0.00000	1,958,304	0
	AND/OR+C	3	81	11	25	122	4,003	37,741	10	0.10607	1,188,441	13,466
pdb1hbk	OR	3	81	36	82	83	0	18,388,104	1,290	0.00000	433,974,365	0
	OR+C	3	81	36	82	83	2,368,665	18,035,836	1,300	0.13133	422,837,769	90,693
	AND/OR	3	81	11	24	83	0	956,908	59	0.00000	23,668,922	0
	AND/OR+C	3	81	11	24	83	56,509	589,861	51	0.09580	16,800,384	162,781
pdb1jer	OR	3	81	33	95	96	0	9,442,948	385	0.00000	129,004,446	0
	OR+C	3	81	33	95	96	1,158,395	7,703,857	344	0.15037	109,765,683	367,006
	AND/OR	3	81	8	18	96	0	26,980	4	0.00000	270,705	0
	AND/OR+C	3	81	8	18	96	1,333	9,996	3	0.13335	121,055	4,489
pdb3c2c	OR	3	81	45	88	89	0	159,754,903	4,491	0.00000	1,167,808,588	0
	OR+C	3	81	45	88	89	37,734	159,754,868	4,575	0.00024	1,167,808,304	2
	AND/OR	3	81	14	25	89	0	97,310,221	4,700	0.00000	2,866,424,518	0
	AND/OR+C	3	81	14	25	89	689,444	26,991,637	977	0.02554	433,467,660	13,056,700

Table 1: Results on pedigree, grid, and pdb instances.

a width by a factor between 2 and 3 (e.g., 55 to 20, or from 63 to 21). Since the proportion of cache nodes is far more significant and since the actual search visits a larger portion of the context-minimal graph the number of cached nodes is far larger for the AND/OR case. Notice that the results we showed were for i-bound of 10. When we used stronger heuristics we observed similar results, yet the pruning of the search space was larger, yielding in most cases a smaller number of nodes expanded and smaller ratios of the number of nodes cached with the number of nodes expanded, making the OR expanded search space even closer to a tree, as the i-bound increases.

Protein (pdb) We observe many instances where the number of caches is high even for OR spaces and the number of cache hits is much larger than in the other domains. This can be explained in part due to the much smaller path-widths in this benchmark—the path-widths vary between 35-50. Thus the underlying context-minimal search graphs contains tree of moderate sizes (e.g. $k^{42}, k^{39}, k^{47}, k^{36}$). Yet, the number of variables is also smaller so the fraction of active caches may not be that large either. We observe also that in this case the search algorithm expands a larger fraction of the context-minimal graph and therefore may see more hit caches. However, caching only marginally reduces the number of nodes expanded in the OR spaces. By contrast, major reductions are observed in the AND/OR spaces.

Conclusion

Our results suggest that the phenomenon we observe can be explained by a combination of hypotheses **H1** and **H2**. That is, due to its higher path-width, the OR search space is several orders of magnitudes larger than the corresponding AND/OR search graph. While OR context-minimal search spaces can have a significant number of active caches, their proportion in the search space is still very small. It is therefore far more likely that DFBnB, who must prune a huge portion of the search space to determine optimality, will not see a particular node, and even more unlikely that it will see a particular node more than once, when searching exponential spaces having a small fraction of active cache nodes.

References

Bodlaender, H. L. 2007. Treewidth: Structure and algorithms. In *Structural Information and Communication Complexity, 14th International Colloquium, SIROCCO 2007, Castiglioncello, Italy, June 5-8, 2007, Proceedings*, 11–25.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 125(1-2):5–41.

Dechter, R., and Mateescu, R. 2007. AND/OR search spaces for graphical models. *Artificial Intelligence* 171(2-3):73–106.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme of approximating inference. *Journal of ACM* 50(2):107–153.

Dechter, R. 2013. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Ihler, A.; Flerova, N.; Dechter, R.; and Otten, L. 2012. Join-graph based cost-shifting schemes. In *Uncertainty in Artificial Intelligence (UAI)*, 397–406.

Kask, K., and Dechter, R. 2001. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence*.

Marinescu, R., and Dechter, R. 2009. Memory intensive and/or search for combinatorial optimization in graphical models. *Artificial Intelligence* 173(16-17):1492–1524.