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
Dechter and Rish (2003).

Since the upper bound at \( n \) is the best solution at \( B \), the upper bound on the solution to the subproblem below.

Consider the example in Fig. 1d where the current lower bound on the optimal solution cost. A AND/OR graph in a depth-first manner while keeping track.

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.

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 \( T \) and if \( \text{context}(X) \subseteq \text{context}(Y) \) then \( \text{context}(Y) \) 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 \( T \).

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 \( T \), denoted \( N(T) \), and the number of active cache nodes denoted, \( C(T) \), obey, \( N(T) = \sum_{X_i \in X} k^{|\text{context}(X_i)|+1} \) \( C(T) = \sum_{\text{active}(X_i) \in X} k^{|\text{context}(X_i)|+1} \). We get the ratio \( R(T) = \frac{C(T)}{N(T)} \).

Computing \( N(T) \) and \( C(T) \) 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 \( T \) (Definitions 1 and 2) we get:

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

**Definition 2** (Induced width, path width) Given a primal graph \( G \) and a pseudo tree \( T \) of a graphical model, the induced width of \( G \) relative to \( T \) 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).
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 $T$ 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 $T$ of $G$ with its associated context-based join-tree, $T$, the number of active cache variables along $T$ 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 $T$ is a pseudo-tree with tree-width $w$ and if $L$ is a pseudo-chain created by depth-first search of $T$, having path-width $pw$, then: (1) The context of every variable in $L$, contains or equals its context in $T$. Consequently; (2) The size of each context may only increase, implying $pw \geq w$; (3) The number of active cache variables in $L$ is smaller or equal to that in $T$; (4) Since $pw \geq w$ we are guaranteed having $pw$ dead-caches in $L$ which is more than the guaranteed dead-caches in $T$.

**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”).

$R(T)$ 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 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. \( k^{42}, k^{30}, k^{17}, k^{10} \)). 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.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Space</th>
<th>( i )</th>
<th>( k )</th>
<th>( w )</th>
<th>( h )</th>
<th>( n )</th>
<th>CACHED</th>
<th>EXPANDED</th>
<th>Time</th>
<th>Ratio</th>
<th>Pruned</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>pedigree</td>
<td>OR</td>
<td>13</td>
<td>4</td>
<td>112</td>
<td>297</td>
<td>298</td>
<td>0</td>
<td>5,762</td>
<td>2</td>
<td>0.00000</td>
<td>5,865</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>4</td>
<td>134</td>
<td>297</td>
<td>297</td>
<td>1,072</td>
<td>5,534</td>
<td>2</td>
<td>0.00000</td>
<td>5,786</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>4</td>
<td>15</td>
<td>144</td>
<td>298</td>
<td>29</td>
<td>2,480</td>
<td>2</td>
<td>0.00000</td>
<td>2,537</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>4</td>
<td>15</td>
<td>144</td>
<td>298</td>
<td>515</td>
<td>990</td>
<td>2</td>
<td>0.05200</td>
<td>1,059</td>
<td>143</td>
</tr>
<tr>
<td>pedigree23</td>
<td>OR+C</td>
<td>13</td>
<td>5</td>
<td>90</td>
<td>308</td>
<td>309</td>
<td>356</td>
<td>191,815</td>
<td>12</td>
<td>0.00000</td>
<td>353,672</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>5</td>
<td>25</td>
<td>52</td>
<td>309</td>
<td>0</td>
<td>330,834</td>
<td>12</td>
<td>0.00000</td>
<td>343,067</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>5</td>
<td>25</td>
<td>52</td>
<td>309</td>
<td>364</td>
<td>98,579</td>
<td>11</td>
<td>0.00000</td>
<td>124,715</td>
<td>27,958</td>
</tr>
<tr>
<td>pedigree37</td>
<td>OR+C</td>
<td>13</td>
<td>5</td>
<td>232</td>
<td>725</td>
<td>726</td>
<td>0</td>
<td>414,737</td>
<td>37</td>
<td>0.00000</td>
<td>570,523</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>5</td>
<td>232</td>
<td>725</td>
<td>726</td>
<td>802</td>
<td>414,491</td>
<td>37</td>
<td>0.00193</td>
<td>570,340</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>5</td>
<td>21</td>
<td>56</td>
<td>726</td>
<td>11,584</td>
<td>184,039</td>
<td>33</td>
<td>0.00000</td>
<td>466,898</td>
<td>33,214</td>
</tr>
<tr>
<td>pedigree39</td>
<td>OR+C</td>
<td>13</td>
<td>5</td>
<td>294</td>
<td>952</td>
<td>953</td>
<td>0</td>
<td>1,149,622,319</td>
<td>21,117</td>
<td>0.00000</td>
<td>1,148,395,317</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AND/OR+C</td>
<td>13</td>
<td>5</td>
<td>294</td>
<td>952</td>
<td>953</td>
<td>2,718</td>
<td>1,149,619,259</td>
<td>21,167</td>
<td>0.00000</td>
<td>1,148,392,739</td>
<td>66</td>
</tr>
</tbody>
</table>

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


