AND/OR Graph Search for Genetic Linkage Analysis

Radu Marinescu and **Rina Dechter**
School of Information and Computer Science
University of California, Irvine, CA 92697-3425
{radum,dechter}@ics.uci.edu

Abstract

AND/OR search spaces have recently been introduced as a unifying framework for advanced algorithmic schemes for graphical models. The main virtue of this representation is its sensitivity to the structure of the model, which can translate into exponential time savings for search algorithms. AND/OR Branch-and-Bound (AOBB) is a new algorithm that explores the AND/OR search tree for solving optimization tasks in graphical models. In this paper we extend the algorithm to explore an AND/OR search *graph* by equipping it with a context-based adaptive caching scheme similar to good and no-good recording. The efficiency of the new graph search algorithm is demonstrated empirically on the very challenging benchmarks that arise in genetic linkage analysis.

Introduction

Graphical models such as belief networks or constraint networks are a widely used representation framework for reasoning with probabilistic and deterministic information. These models use graphs to capture conditional independencies between variables, allowing a concise representation of the knowledge as well as efficient graph-based query processing algorithms. Optimization tasks such as finding the most likely state of a belief network or finding a solution that violates the least number of constraints can be defined within this framework and they are typically tackled with either *search* or *inference* algorithms (Dechter 2003).

The AND/OR search space for graphical models (Dechter & Mateescu 2006) is a new framework for search that is sensitive to the independencies in the model, often resulting in exponentially reduced complexities. It is based on a pseudo-tree that captures independencies in the graphical model, resulting in a search tree exponential in the depth of the pseudo-tree, rather than in the number of variables.

AND/OR Branch-and-Bound algorithm (AOBB) is a new search method that explores the AND/OR search tree for solving optimization tasks in graphical models (Marinescu & Dechter 2005). In this paper we improve the AOBB scheme significantly by using *caching* schemes. Namely, we extend the algorithm to explore the AND/OR search *graph* rather than the AND/OR search tree, using a flexible caching mechanism that can adapt to memory limitations.

The caching scheme is based on *contexts* and is similar to good and no-good recording and recent schemes appearing in Recursive Conditioning (Darwiche 2001) and Valued Backtracking (Bacchus, Dalmao, & Pittasi 2003). The efficiency of the proposed search methods also depends on the accuracy of the guiding heuristic function, which is based on the mini-bucket approximation of Variable Elimination (Dechter & Rish 2003). We focus our empirical evaluation on the task of finding the Most Probable Explanation in belief networks (Pearl 1988), and illustrate our results on several benchmarks from the field of genetic linkage analysis.

The paper is organized as follows. Section 2 provides background on belief networks, AND/OR search trees and the AOBB algorithm. In Section 3 we introduce the AND/OR search *graph* and AOBB with caching. In Section 4 we describe two context-based caching schemes. Section 5 gives some experimental results and Section 6 concludes.

Preliminaries

Belief Networks

Belief Networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are defined by a directed acyclic graph over nodes representing variables of interest.

DEFINITION **1 (belief network)** *A* belief network *is a quadruple $\mathcal{B} = (\mathcal{X}, \mathcal{D}, \mathcal{G}, \mathcal{P})$, where $\mathcal{X} = \{X_1, ..., X_n\}$ is a set of random variables, $\mathcal{D} = \{D_1, ..., D_n\}$ is the set of the corresponding discrete-valued domains, $\mathcal{G}$ is a directed acyclic graph over $\mathcal{X}$ and $\mathcal{P} = \{p_1, ..., p_n\}$, where $p_i = P(X_i|pa(X_i))$ ($pa(X_i)$ are the parents of X_i in $\mathcal{G}$) denote* conditional probability tables *(CPTs). The belief network represents a joint probability distribution over $\mathcal{X}$ having the product form $P_{\mathcal{B}}(\bar{x}) = \prod_{i=1}^{n} P(x_i|x_{pa_i})$, where an assignment $(X_1 = x_1, ..., X_n = x_n)$ is abbreviated to $\bar{x} = (x_1, ..., x_n)$ and where x_S denotes the restriction of a tuple x over a subset of variables S. An evidence set e is an instantiated subset of variables. The* moral graph *of a belief network is the undirected graph obtained by connecting the parent nodes of each variable and eliminating direction.*

The primary optimization query over belief networks is finding the *Most Probable Explanation* (MPE), namely,

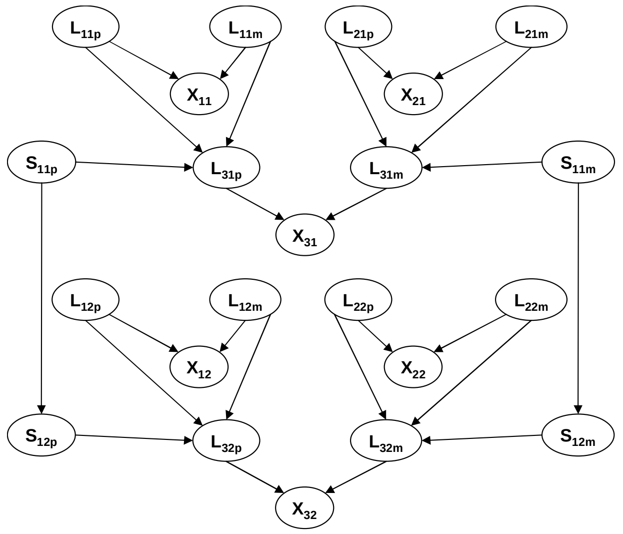


Figure 1: A fragment of a belief network used in genetic linkage analysis.

finding a complete assignment to all variables having maximum probability, given the evidence. A generalization of the MPE query is *Maximum a Posteriori Hypothesis* (MAP), which calls for finding the most likely assignment to a subset of hypothesis variables, given the evidence.

DEFINITION **2 (MPE task)** *Given a belief network and evidence* e, *the* Most Probable Explanation (MPE) *task is to find an assignment* $(x_1^o, ..., x_n^o)$ *such that:* $P(x_1^o, ..., x_n^o) = max_{X_1,...,X_n} \prod_{k=1}^n P(X_k|pa(X_k), e)$.

The MPE task appears in applications such as diagnosis, abduction and explanation. For example, given data on clinical findings, MPE can postulate on a patient's probable afflictions. In decoding, the task is to identify the most likely message transmitted over a noisy channel given the observed output.

DEFINITION **3 (induced graph, induced width)** *Given a graph* G, *its* induced graph *relative to an ordering* d *of the variables, denoted* $G^*(d)$, *is obtained by processing the nodes in reverse order of* d. *For each node all its earlier neighbors are connected, including neighbors connected by previously added edges. Given a graph and an ordering of its nodes, the* width *of a node is the number of edges connecting it to nodes lower in the ordering. The* induced width *of a graph, denoted* $w^*(d)$, *is the maximum width of nodes in the induced graph.*

Genetic Linkage Analysis

In human genetic linkage analysis (Ott 1999), the *haplotype* is the sequence of alleles at different loci inherited by an individual from one parent, and the two haplotypes (maternal and paternal) of an individual constitute this individual's *genotype*. When genotypes are measured by standard procedures, the result is a list of unordered pairs of alleles, one pair for each locus. The *maximum likelihood haplotype* problem consists of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data.

The pedigree data can be represented as a belief network with three types of random variables: *genetic loci* variables which represent the genotypes of the individuals in the pedigree (two genetic loci variables per individual per locus, one for the paternal allele and one for the maternal allele), *phenotype* variables, and *selector* variables which are auxiliary variables used to represent the gene flow in the pedigree. Figure 1 represents a fragment of a network that describes parents-child interactions in a simple 2-loci analysis. The genetic loci variables of individual i at locus j are denoted by $L_{i,jp}$ and $L_{i,jm}$. Variables $X_{i,j}$, $S_{i,jp}$ and $S_{i,jm}$ denote the phenotype variable, the paternal selector variable and the maternal selector variable of individual i at locus j, respectively. The conditional probability tables that correspond to the selector variables are parameterized by the *recombination ratio* θ (Fishelson & Geiger 2002). The remaining tables contain only deterministic information. It can be shown that given the pedigree data, the haplotyping problem is equivalent to computing the Most Probable Explanation (MPE) of the corresponding belief network (for more details consult (Fishelson & Geiger 2002; Fishelson, Dovgolevsky, & Geiger 2005)).

AND/OR Search Trees

The usual way to do search is to instantiate variables in turn (we only consider a static variable ordering). In the simplest case, this process defines a search tree (called here OR search tree), whose nodes represent states in the space of partial assignments. The traditional search space does not capture the structure of the underlying graphical model. Introducing AND states into the search space can capture the structure decomposing the problem into independent subproblems by conditioning on values (Freuder & Quinn 1985; Dechter & Mateescu 2006). The AND/OR search space is defined using a backbone *pseudo-tree*.

DEFINITION **4 (pseudo-tree)** *Given an undirected graph* $G = (V, E)$, *a directed rooted tree* $T = (V, E')$ *defined on all its nodes is called* pseudo-tree *if any arc of* G *which is not included in* E' *is a back-arc, namely it connects a node to an ancestor in* T.

Given a belief network $\mathcal{B} = (\mathcal{X}, \mathcal{D}, \mathcal{P})$, its moral graph G and a pseudo-tree T of G, the associated AND/OR search tree S_T has alternating levels of OR nodes and AND nodes. The OR nodes are labeled X_i and correspond to the variables. The AND nodes are labeled $\langle X_i, x_i \rangle$ and correspond to value assignments in the domains of the variables. The structure of the AND/OR tree is based on the underlying pseudo-tree arrangement T of G. The root of the AND/OR search tree is an OR node, labeled with the root of T.

The children of an OR node X_i are AND nodes labeled with assignments $\langle X_i, x_i \rangle$, consistent along the path from the root, $path(X_i, x_i) = (\langle X_1, x_1 \rangle, ..., \langle X_{i-1}, x_{i-1} \rangle)$. The children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labeled with the children of variable X_i in T. In other words, the OR states represent alternative ways of solving the problem,

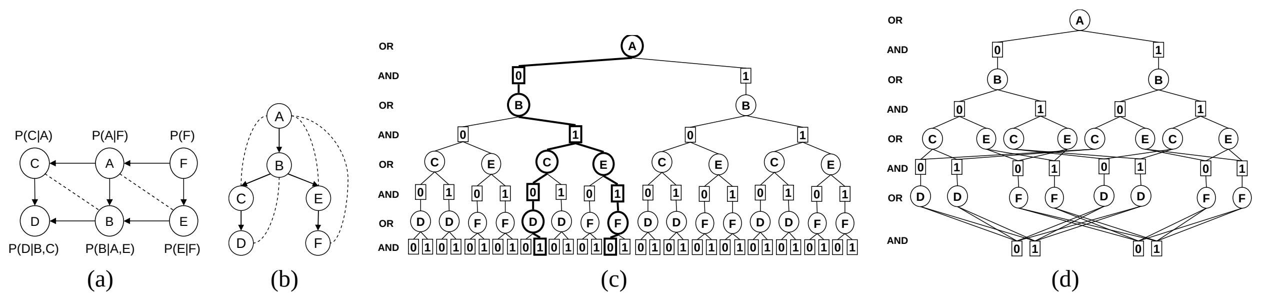


Figure 2: AND/OR search spaces

whereas the AND states represent problem decomposition into independent subproblems, all of which need be solved. When the pseudo-tree is a chain, the AND/OR search tree coincides with the regular OR search tree.

A *solution subtree* Sol_{S_T} of S_T is an AND/OR subtree such that: (i) it contains the root of S_T; (ii) if a nonterminal AND node $n \in S_T$ is in Sol_{S_T} then all its children are in Sol_{S_T}; (iii) if a nonterminal OR node $n \in S_T$ is in Sol_T then exactly one of its children is in Sol_{S_T}.

Example 1 *Figures 2(a) and 2(b) show a belief network and its pseudo-tree together with the back-arcs (dotted lines). Figure 2(c) shows the AND/OR search tree based on the pseudo-tree, for bi-valued variables. A solution subtree is highlighted.*

The AND/OR search tree can be traversed by a depth-first search algorithm that is guaranteed to have a time complexity exponential in the depth of the pseudo-tree and can use linear space (Dechter & Mateescu 2006). The arcs from X_i to $\langle X_i, x_i\rangle$ are annotated by appropriate *labels* of the functions in $\mathcal{P}$. The nodes in S_T can be associated with *values*, defined over the subtrees they root.

DEFINITION **5 (label)** *The* label $l(X_i, x_i)$ *of the arc from the OR node X_i to the AND node $\langle X_i, x_i\rangle$ is defined as the product of all the conditional probability tables whose scope includes X_i and is fully assigned along $path(X_i, x_i)$.*

DEFINITION **6 (value)** *The* value $v(n)$ *of a node $n \in S_T$ is defined recursively as follows: (i) if $n = \langle X_i, x_i\rangle$ is a terminal AND node then $v(n) = l(X_i, x_i)$; (ii) if $n = \langle X_i, x_i\rangle$ is an internal AND node then $v(n) = l(X_i, x_i) \cdot \prod_{n' \in succ(n)} v(n')$; (iii) if $n = X_i$ is an internal OR node then $v(n) = max_{n' \in succ(n)} v(n')$, where $succ(n)$ are the children of n in S_T.*

Clearly, the value of each node can be computed recursively, from leaves to root.

PROPOSITION **1** *Given an AND/OR search tree S_T of a belief network $\mathcal{B} = (\mathcal{X}, \mathcal{D}, \mathcal{P})$, the value $v(n)$ of a node $n \in S_T$ is the most probable explanation of the subproblem rooted at n, subject to the current variable instantiation along the path from root to n. If n is the root of S_T, then $v(n)$ is the most probable explanation of $\mathcal{B}$.*

AND/OR Branch-and-Bound Tree Search

AND/OR Branch-and-Bound (AOBB) was introduced in (Marinescu & Dechter 2005) as a depth-first Branch-and-Bound that explores an AND/OR search tree for solving optimization tasks in graphical models. In the following we review briefly the algorithm.

At any stage during search, a node n along the current path roots a current *partial solution subtree*, denoted by $S_{sol}(n)$, which must be connected, must contain its root n and will have a *frontier* containing all those nodes that were generated and not yet expanded. Furthermore, there exists a *static* heuristic function $h(n)$ overestimating $v(n)$ that can be computed efficiently when node n is first generated.

Given the current partially explored AND/OR search tree S_T, the *active path* $\mathcal{AP}(t)$ is the path of assignments from the root of S_T to the current tip node t. The *inside context* $in(\mathcal{AP})$ of $\mathcal{AP}(t)$ contains all nodes that were fully evaluated and are children of nodes on $\mathcal{AP}(t)$. The *outside context* $out(\mathcal{AP})$ of $\mathcal{AP}(t)$, contains all the frontier nodes that are children of the nodes on $\mathcal{AP}(t)$. The *active partial subtree* $\mathcal{APT}(n)$ rooted at a node $n \in \mathcal{AP}(t)$ is the subtree of $S_{sol}(n)$ containing the nodes on $\mathcal{AP}(t)$ between n and t together with their OR children. A *dynamic heuristic evaluation function* of a node n relative to $\mathcal{APT}(n)$ which overestimates $v(n)$ is defined as follows (for more details see (Marinescu & Dechter 2005)).

DEFINITION **7 (dynamic heuristic evaluation function)** *Given an active partial tree $\mathcal{APT}(n)$, the* dynamic heuristic evaluation function *of n, $f_h(n)$, is defined recursively as follows: (i) if $\mathcal{APT}(n)$ consists only of a single node n, and if $n \in in(\mathcal{AP})$ then $f_h(n) = v(n)$ else $f_h(n) = h(n)$; (ii) if $n = \langle X_i, x_i\rangle$ is an AND node, having OR children $m_1, ..., m_k$ then $f_h(n) = min(h(n), l(X_i, x_i) \cdot \prod_{i=1}^{k} f_h(m_i))$; (iii) if $n = X_i$ is an OR node, having an AND child m, then $f_h(n) = min(h(n), f_h(m))$.*

AOBB traverses the AND/OR search tree in a depth-first manner and calculates an *upper bound* on $v(n)$ of any node n on the active path, by using $f_h(n)$. It also maintains an *lower bound* on $v(n)$ which is the current best solution subtree rooted at n. If $f_h(n) \leq lb(n)$ then the search is terminated below the tip node of the active path.

AND/OR Search Graphs

The AND/OR search tree may contain nodes that root identical subtrees (i.e. their root nodes values are identical). These are called *unifiable*. When unifiable nodes are merged, the search tree becomes a graph and its size becomes smaller. A depth-first search algorithm can explore the AND/OR graph using additional memory. The algorithm can 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*.

DEFINITION **8 (context)** *Given a belief network and the corresponding AND/OR search tree S_T relative to a pseudo-tree T, the* context *of any AND node $\langle X_i, x_i \rangle \in S_T$, denoted by $context(X_i)$, is defined as the set of ancestors of X_i in T, including X_i, that are connected to descendants of X_i.*

It is easy to verify that the context of X_i d-separates (Pearl 1988) the subproblem P_{X_i} below X_i from the rest of the network. Namely, it is possible to solve P_{X_i} for any assignment of $context(X_i)$ and record its optimal value, thus avoiding to solve P_{X_i} again for the same assignment. The *context-minimal* AND/OR graph is obtained by merging all the context unifiable AND nodes. The size of the largest context is bounded by the induced width w^* of the moral graph (extended with the pseudo-tree extra arcs) over the ordering given by the depth-first traversal of T (i.e. induced width of the pseudo-tree). Therefore, the time and space complexity of a search algorithm traversing the context-minimal AND/OR graph is $O(exp(w^*))$ (Dechter & Mateescu 2006).

For illustration, consider the context-minimal graph in Figure 2(d) of the pseudo-tree from Figure 2(b). Its size is far smaller that that of the AND/OR tree from Figure 2(c) (16 nodes vs. 54 nodes). The contexts of the nodes can be read from the pseudo-tree, as follows: $context(A)$ = {A}, $context(B)$ = {B,A}, $context(C)$ = {C,B}, $context(D)$ = {D}, $context(E)$ = {E,A} and $context(F)$ = {F}.

AND/OR Branch-and-Bound Graph Search

In this section we extend AOBB to traverse an AND/OR search graph by equipping it with a caching mechanism.

Figure 3 shows the graph AOBB$_g$ algorithm. The following notation is used: $(\mathcal{X}, \mathcal{D}, \mathcal{P})$ is the problem with which the procedure is called, st is the current partial solution subtree being explored, in (resp. out) is the inside (resp. outside) context of the active path. The algorithm assumes that variables are selected according to a pseudo-tree.

If the set $\mathcal{X}$ is empty, then the result is trivially computed (line 1). Else, AOBB$_g$ selects a variable X_i (i.e. expands the OR node X_i) and iterates over its values (line 5) to compute the OR value $v(X_i)$. The algorithm attempts to retrieve the results cached at the AND nodes (line 7). If a valid cache entry v is found for the current AND node $\langle X_i, x_i \rangle$ then the OR value $v(X_i)$ is updated (line 11) and the search continues with the next value in X_i's domain. Otherwise, the problem is decomposed into a set of q independent subproblems, one for each child X_k of X_i in the pseudo-tree. Procedure UB computes the static heuristic function $h(n)$ for every node in the search tree.

```
function: AOBB_g(st, $\mathcal{X}$, $\mathcal{D}$, $\mathcal{P}$)
1  if $\mathcal{X} = \emptyset$ then return 0;
2  else
3      $X_i \leftarrow$ SelectVar($\mathcal{X}$);
4      $v(X_i) \leftarrow 0$;
5      foreach $x_i \in D_i$ do
6          $st' \leftarrow st \cup (X_i, x_i)$;
7          $v \leftarrow$ ReadCache($X_i$,$x_i$);
8          if $v \neq$ NULL then
9              $tmp \leftarrow v \cdot$ label($X_i$,$x_i$);
10             if ¬FindCut($X_i$,$x_i$,in,out,$tmp$) then
11                 $v(X_i) \leftarrow max(v(X_i), tmp)$;
12             continue;
13         end
14         $h(X_i, x_i) \leftarrow$ UB($\mathcal{X}$,$\mathcal{D}$,$\mathcal{P}$);
15         foreach $k = 1..q$ do
16             $h(X_k) \leftarrow$ UB($\mathcal{X}_k$,$\mathcal{D}_k$,$\mathcal{P}_k$);
17             UpdateContext(out, $X_k$, $h(X_k)$);
18         end
19         if ¬FindCut($X_i$,$x_i$,in,out,$h(X_i, x_i)$) then
20             $v(X_i, x_i) \leftarrow 1$;
21             foreach $k = 1..q$ do
22                 $val \leftarrow$AOBB_g($st'$,$\mathcal{X}_k$,$\mathcal{D}_k$,$\mathcal{P}_k$);
23                 $v(X_i, x_i) \leftarrow v(X_i, x_i) \cdot val$;
24             end
25             WriteCache($X_i$,$v(X_i, x_i)$);
26             $v(X_i, x_i) \leftarrow v(X_i, x_i) \cdot$label($X_i$,$x_i$);
27             UpdateContext(in, $v(X_i, x_i)$);
28             $v(X_i) \leftarrow max(v(X_i), v(X_i, x_i))$;
29         end
30     end
31     return $v(X_i)$;
32 end
```

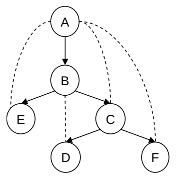
Figure 3: Graph AND/OR Branch-and-Bound.

When expanding the AND node $\langle X_i, x_i \rangle$, AOBB$_g$ successively updates the *dynamic heuristic function* $f_h(m)$ for every ancestor node m along the active path and terminates the current search path if, for some m, $f_h(m) \leq lb(m)$. Else, the independent subproblems are sequentially solved (line 21) and the solutions are accumulated by the AND value $v(X_i, x_i)$ (line 23). After trying all feasible values of variable X_i, the most probable solution to the subproblem rooted by X_i remains in $v(X_i)$, which is returned (line 31).

The Mini-Bucket Heuristics

In this section we describe briefly a general scheme for generating static heuristic estimates $h(n)$, based on the Mini-Bucket approximation. The scheme is parameterized by the Mini-Bucket i-bound, which allows for a controllable trade-off between heuristic strength and its overhead.

Mini-Bucket Elimination (MBE) (Dechter & Rish 2003) is an approximation algorithm designed to avoid the high time and space complexity of *Bucket Elimination* (BE) (Dechter 1999), by partitioning large buckets into smaller subsets, called *mini buckets*, each containing at most i (called i-bound) distinct variables. The mini-buckets are then processed separately. The algorithm outputs not only a bound on the optimal solution cost, but also the collection of augmented buckets, which form the basis for the heuristics gen-



(a)

B(F): [P(F|A,C)]

B(D): [P(D|B,C)]

B(C): [P(C|A) || λ^F(A,C)], [λ^D(B,C)]

B(E): [P(E|A,B)]

B(B): [P(B|A) || λ^E(A,B), λ^C(B)]

B(A): [P(A) || λ^B(A), λ^C(A)]

(b)

Figure 4: Schematic execution of MBE(2).

erated. The complexity is time and space $O(exp(i))$.

In the past, (Kask & Dechter 2001) showed that the intermediate functions generated by the Mini-Bucket algorithm MBE(i) can be used to compute a heuristic function, that overestimates the most probable extension of the current partial assignment in a regular OR search tree. More recently, (Marinescu & Dechter 2005) extended the idea to AND/OR search spaces as well.

Assume that a belief network $\mathcal{B} = (\mathcal{X}, \mathcal{D}, \mathcal{P})$ with pseudo-tree T is being solved by AOBB search, where the active path ends with some OR node X_j. Consider also the *augmented* bucket structure $\{B(X_1), ..., B(X_n)\}$ of $\mathcal{B}$, constructed along the ordering resulted from a depth-first traversal of T. For each possible value assignment $X_j = x_j$, the *static mini-bucket heuristic estimate* $h(x_j)$ of the most probable solution rooted by X_j can be computed as the product of the original conditional probability tables in bucket $B(X_j)$ and the intermediate functions λ^k that were generated in buckets $B(X_k)$ and reside in bucket $B(X_j)$ or below, where X_k is a descendant of X_j in T (more details in (Kask & Dechter 2001; Marinescu & Dechter 2005)).

Example 2 *Figure 4(b) shows the augmented bucket structure generated by MBE(i=2) for the pseudo-tree displayed in Figure 4(a), along the ordering* (A, B, E, C, D, F)*; square brackets denote the choice of partitioning. Assume that during search, the active path of the current partial solution subtree is* $(A = a, B = b)$ *and the tip node is the OR node* C*. The static mini-bucket heuristic estimate* $h(C = c) = P(c|a) \cdot \lambda^F(a, c) \cdot \lambda^D(b, c)$.

Caching Schemes

In this section we present two caching schemes that can adapt to the current memory limitations. They are based on *contexts*, which are pre-computed from the pseudo-tree and use a parameter called *cache bound* (or j-bound) to control the amount of memory used for storing unifiable nodes.

Naive Caching

The first scheme, called *naive caching* and denoted by AOBB+C(j), stores nodes at the variables whose context size is smaller than or equal to the cache bound j. It is easy to see that when j equals the induced width of the pseudo-tree the algorithm explores the context-minimal AND/OR graph.

A straightforward way of implementing the caching scheme is to have a *cache table* for each variable X_k recording the context. Specifically, lets assume that the context of X_k is $context(X_k) = \{X_i, ..., X_k\}$ and $|context(X_k)| \leq j$. A cache table entry corresponds to a particular instantiation $\{x_i, ..., x_k\}$ of the variables in $context(X_k)$ and records the most probable solution to the subproblem P_{X_k}.

However, some tables might never get cache hits. We call these *dead-caches*. In the AND/OR search graph, dead-caches appear at nodes that have only one incoming arc. AOBB+C(j) needs to record only nodes that are likely to have additional incoming arcs, and these nodes can be determined by inspecting the pseudo-tree. Namely, if the context of a node includes that of its parent, then there is no need to store anything for that node, because it would be a dead-cache. For example, node B in the AND/OR search graph from Figure 2(c) is a dead-cache because its context includes the context of its parent A in the pseudo-tree.

Adaptive Caching

The second scheme, called *adaptive caching* and denoted by AOBB+AC(j), is inspired by the AND/OR cutset conditioning scheme and was first explored in (Mateescu & Dechter 2005). It extends the naive scheme by allowing caching even at nodes with contexts larger than the given cache bound, based on *adjusted contexts*.

We will illustrate the idea with an example. Consider the node X_k with $context(X_k) = \{X_i, ..., X_k\}$, where $|context(X_k)| > j$. During search, when variables $\{X_i, ..., X_{k-j}\}$ are assigned, they can be viewed as part of a w*-cutset* (Pearl 1988). The w-cutset method consists of enumerating all the possible instantiations of a subset of variables (i.e. cutset), and for each one solving the remaining easier subproblem within w-bounded space restrictions.

Therefore, once variables $\{X_i, ..., X_{k-j}\}$ are instantiated, the problem rooted at X_{k-j+1} can be solved as a simplified subproblem from the cutset method. In the subproblem, conditioned on the values $\{x_i, ..., x_{k-j}\}$, $context(X_k)$ is $\{X_{k-j+1}, ..., X_k\}$ (we call this the *adjusted context* of X_k), so it can be stored within the j-bounded space restrictions. However, when AOBB+AC(j) retracts to X_{k-j} or above, all the nodes cached at variable X_k need to be discarded.

This caching scheme requires only a linear increase in additional memory, compared to AOBB+C(j), but it has the potential of exponential time savings. Specifically, for solving the subproblem rooted by X_k, AOBB+AC(j) requires $O(exp(m))$ time and $O(exp(j))$ space, whereas AOBB+C(j) needs $O(exp(h_k))$ time and linear space, where h_k is the depth of the subtree rooted at X_k in the pseudo-tree, $m = |context(X_k)|$ and $m \leq h_k$.

Additional dead-caches in the adaptive scheme can also be identified by inspecting the pseudo-tree. Consider the node X_k from the previous example and let $anc(X_k)$ be the ancestors of X_k in the pseudo-tree between X_k and X_{k-j}, including X_k. If $anc(X_k)$ contains only the variables in the adjusted context of X_k then X_k is a dead-cache.

Preliminary Experiments

In this section we evaluate empirically the performance of the AND/OR Branch-and-Bound graph search algorithm on

ped	(w*, h)	VEC	SUPERLINK	(i, j)	AOMB(i)		AOMB+C(i,j)		AOMB+AC(i,j)	
					time	nodes	time	nodes	time	nodes
1	(15, 61)	24.62	131.3	(10, 10)	0.609	23,787	0.249	4,723	**0.218**	4,191
20	(24, 69)	1,304	**12.44**	(16, 16)	480.2	19,118,600	182.0	5,072,650	192.0	5,072,400
23	(23, 38)	1,144	6,809	(16, 18)	16.60	382,351	11.33	161,896	**11.29**	159,377
30	(26, 51)	26,719	28,740	(20, 22)	61.57	925,958	38.85	164,701	**38.81**	162,061
38	(17, 59)	15,860	**62.18**	(12, 12)	1,212	35,360,600	104.4	1,206,780	124.7	1,156,140
50	(18, 58)	85,637	716.6	(10, 12)	83.52	2,312,423	**29.72**	445,083	36.41	444,058

Table 1: Time in seconds and nodes visited to prove optimality for genetic linkage analysis.

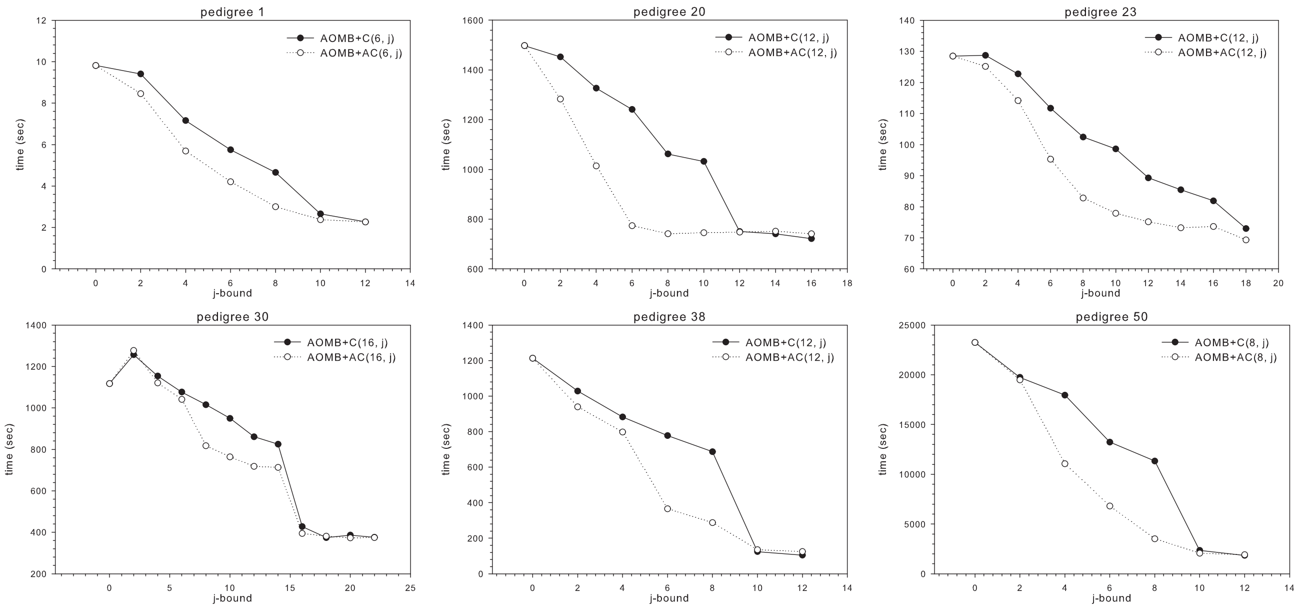


Figure 5: Detailed time results in seconds comparing the naive vs. adaptive caching for genetic linkage analysis.

the task of finding the most likely haplotype configuration of a general pedigree. All our experiments were done on a 2.4GHz Pentium IV with 2GB of RAM.

We consider two classes of AND/OR Branch-and-Bound graph search algorithms guided by the pre-compiled mini-bucket heuristics and using either the *naive* or *adaptive* caching schemes. They are denoted by AOMB+C(i,j) and AOMB+AC(i,j), respectively. The parameters i and j denote the mini-bucket i-bound (which controls the accuracy of the heuristic) and the cache bound. The pseudo-trees were generated using the min-fill heuristic, as described in (Marinescu & Dechter 2005).

We report the average effort as CPU time (in seconds) and number of nodes visited, required for proving optimality of the solution, the induced width (w*) and depth of the pseudo-tree (h) obtained for the test instances. The best performance points are highlighted. For comparison, we also report results obtained with the tree version of the algorithms denoted by AOMB(i). The latter was shown to outperform significantly the OR Branch-and-Bound version (BBMB) in various domains (Marinescu & Dechter 2005).

Table 1 displays a summary of the results obtained for 6 hard linkage analysis networks[1]. For comparison, we include results obtained with VEC and SUPERLINK. SUPERLINK is currently the most efficient solver for genetic linkage analysis, is dedicated to this domain, uses a combination of variable elimination and conditioning, and takes advantage of the determinism in the network. VEC is our implementation of the elimination/conditioning hybrid and is not sensitive to determinism.

We observe that AOMB+C(i,j) and AOMB+AC(i,j) are the best performing algorithms in this domain. The time savings caused by both naive and adaptive caching schemes are significant and in some cases the differences add up to several orders of magnitude over both VEC and SUPERLINK (e.g. ped-23, ped-50). Figure 6 provides an alternative view comparing the two caching schemes, in terms of CPU time, for a smaller i-bound of the mini-bucket heuristic. We notice that adaptive caching improves significantly over the naive scheme especially for relatively small j-bounds. This may be important because small j-bounds mean restricted space. At large j-bounds the two schemes are identical.

In summary, the effect of caching (either naive or adaptive) is more prominent for relatively weak guiding heuris-

[1]http://bioinfo.cs.technion.ac.il/superlink/

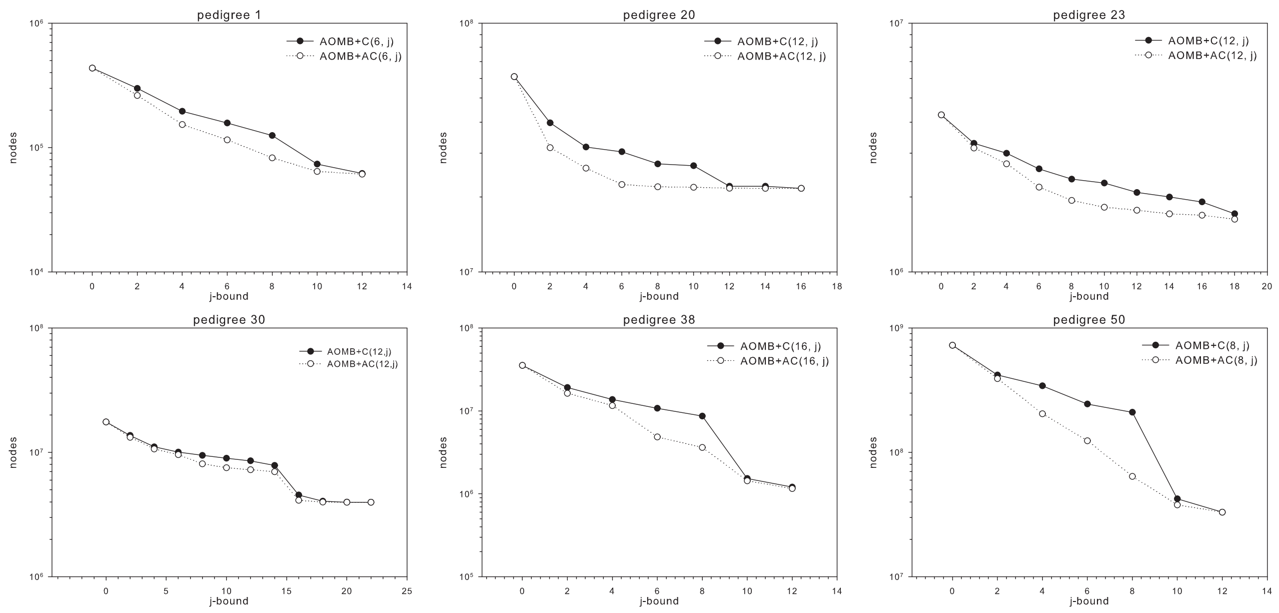


Figure 6: Detailed number of nodes visited comparing the naive vs. adaptive caching for genetic linkage analysis.

tics estimates. The merit of adaptive caching over naive one is evident when the j-bound is much smaller than the induced width and there is a relatively small number of dead-caches. This translates sometimes into impressive time savings for the Branch-and-Bound algorithms.

Conclusion

In this paper we extended the AND/OR Branch-and-Bound algorithm to traversing an AND/OR search graph rather than an AND/OR search tree by equipping it with an efficient caching mechanism. We investigated two flexible context-based caching schemes that can adapt to the current memory restrictions. The efficiency of the new AND/OR Branch-and-Bound graph search algorithms is demonstrated empirically on several challenging benchmarks from the field of genetic linkage analysis.

Related Work: AOBB graph search is related to the Branch-and-Bound method proposed by (Kanal & Kumar 1988) for acyclic AND/OR graphs and game trees, as well as the pseudo-tree search algorithm proposed in (Larrosa, Meseguer, & Sanchez 2002). BTD developed in (Jegou & Terrioux 2004) can also be interpreted as an AND/OR graph search algorithm with a caching mechanism based on the separators of the guiding tree-decomposition.

References

Bacchus, F.; Dalmao, S.; and Pittasi, T. 2003. Value elimination: Bayesian inference via backtracking search. *In Uncertainty in Artificial Intelligence (UAI'03)* 20–28.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 126(1-2):5–41.

Dechter, R., and Mateescu, R. 2006. And/or search spaces for graphical models. *UCI-ICS Technical Report*.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme for approximating inference. *Journal of ACM*.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*.

Dechter, R. 2003. *Constraint Processing*. MIT Press.

Fishelson, M., and Geiger, D. 2002. Exact genetic linkage computations for general pedigrees. *Bioinformatics*.

Fishelson, M.; Dovgolevsky, N.; and Geiger, D. 2005. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*.

Freuder, E., and Quinn, M. 1985. Taking advantage of stable sets of variables in csps. *In IJCAI'85* 1076–1078.

Jegou, P., and Terrioux, C. 2004. Decomposition and good recording for solving max-csps. *In ECAI'04* 196–200.

Kanal, L., and Kumar, V. 1988. *Search in artificial intelligence*. Springer-Verlag.

Kask, K., and Dechter, R. 2001. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence* 129:91–131.

Larrosa, J.; Meseguer, P.; and Sanchez, M. 2002. Pseudo-tree search with soft constraints. *In ECAI'02* 131–135.

Marinescu, R., and Dechter, R. 2005. And/or branch-and-bound for graphical models. *In IJCAI'05* 224–229.

Mateescu, R., and Dechter, R. 2005. And/or cutset conditioning. *In IJCAI'05* 230–235.

Ott, J. 1999. *Analysis of Human Genetic Linkage*. The Johns Hopkins University Press.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann.