# 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,

Copyright © 2007, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

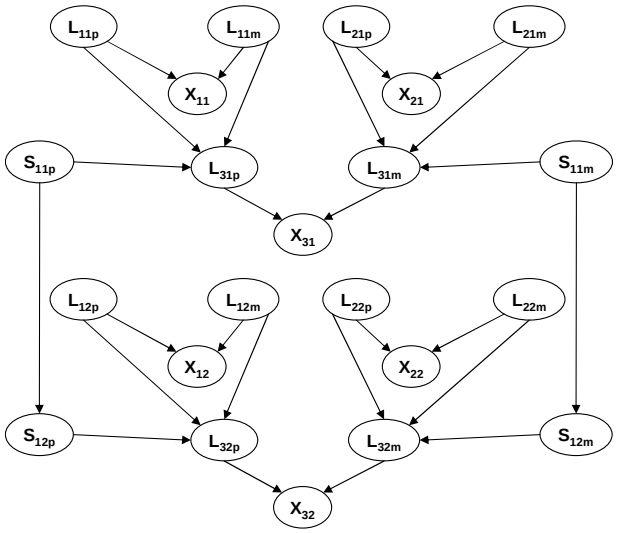



Figure 1: A fragment of a belief network used in genetic linkage analysis.

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* $\theta$ (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,

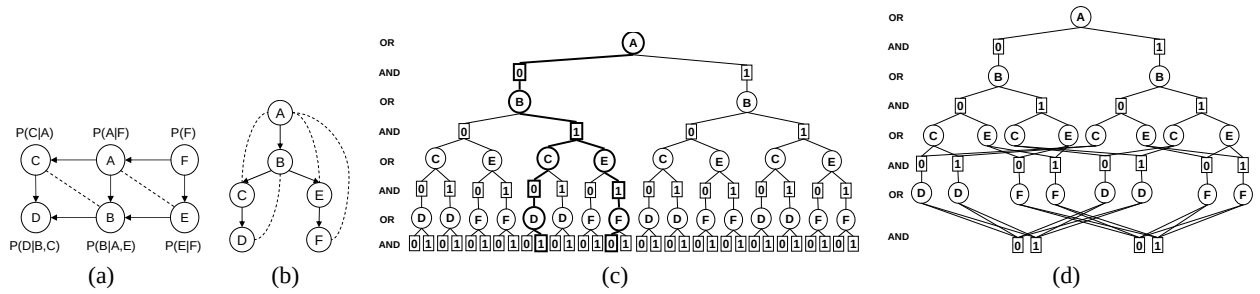



Figure 2: AND/OR search spaces

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 1 shows the graph $\mathsf{AOBB}_g$ 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, $\mathsf{AOBB}_g$ 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.

**Algorithm 1**: Graph AND/OR Branch-and-Bound.

```
function: AOBB_g(st, X, D, P)
1  if X = ∅ then return 0;
2  else
3      X_i ← SelectVar(X);
4      v(X_i) ← 0;
5      foreach x_i ∈ D_i do
6          st' ← st ∪ (X_i, x_i);
7          v ← ReadCache(X_i,x_i);
8          if v ≠ NULL then
9              tmp ← v · label(X_i,x_i);
10             if ¬FindCut(X_i,x_i,in,out,tmp) then
11                 v(X_i) ← max(v(X_i), tmp);
12             continue;
13         h(X_i, x_i) ← UB(X, D, P);
14         foreach k = 1..q do
15             h(X_k) ← UB(X_k,D_k,P_k);
16             UpdateContext(out, X_k, h(X_k));
17         if ¬FindCut(X_i,x_i,in,out,h(X_i, x_i)) then
18             v(X_i, x_i) ← 1;
19             foreach k = 1..q do
20                 val ← AOBB_g(st',X_k,D_k,P_k);
21                 v(X_i, x_i) ← v(X_i, x_i) · val;
22             WriteCache(X_i,v(X_i, x_i));
23             v(X_i, x_i) ← v(X_i, x_i)·label(X_i,x_i);
24             UpdateContext(in, v(X_i, x_i));
25             v(X_i) ← max(v(X_i), v(X_i, x_i));
26     return v(X_i);
```

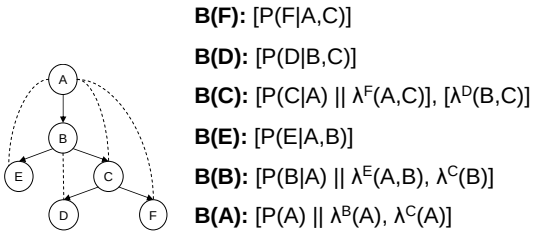
When expanding the AND node $\langle X_i, x_i \rangle$, $\mathsf{AOBB}_g$ successively updates the *dynamic heuristic function* $f_h(m)$ for every ancestor node $m$ along the active path and terminates the current search path if, for some $m$, $f_h(m) \leq lb(m)$. Else, the independent subproblems are 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 generated. 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




(a) (b)

Figure 3: Schematic execution of MBE(2).

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 $\lambda^k$ 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 3(b) shows the augmented bucket structure generated by MBE(i=2) for the pseudo-tree displayed in Figure 3(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 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.

| 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.

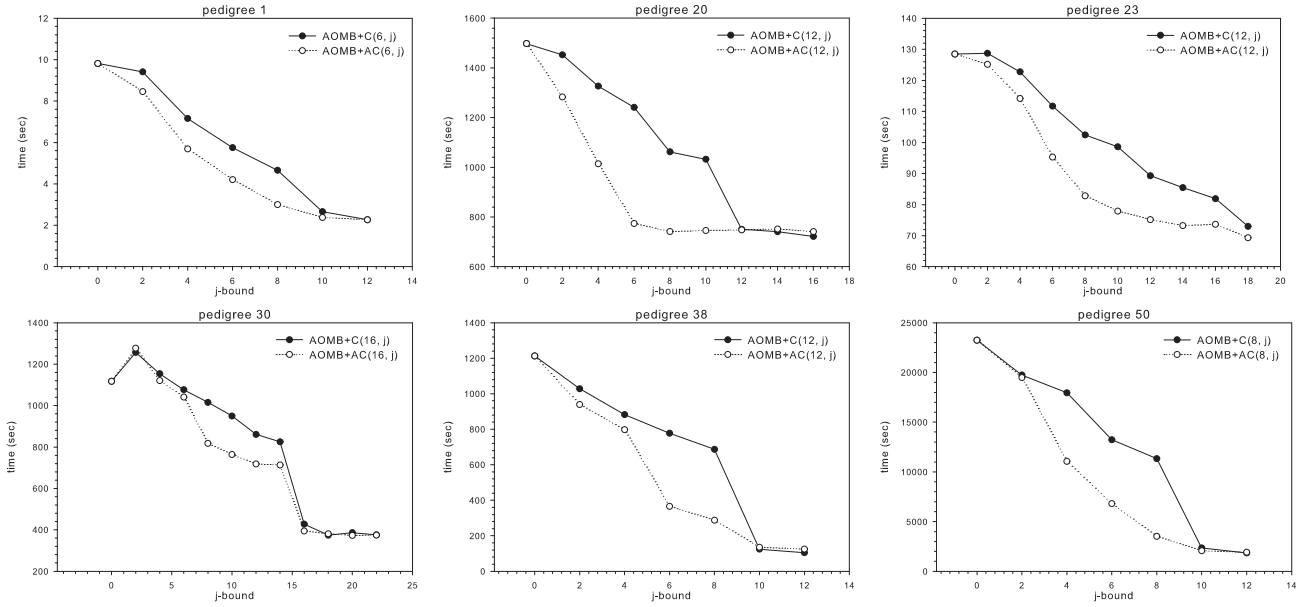



Figure 4: Detailed time results in seconds comparing the naive vs. adaptive caching for genetic linkage analysis.

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 5 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 heuristics 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 sav-

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

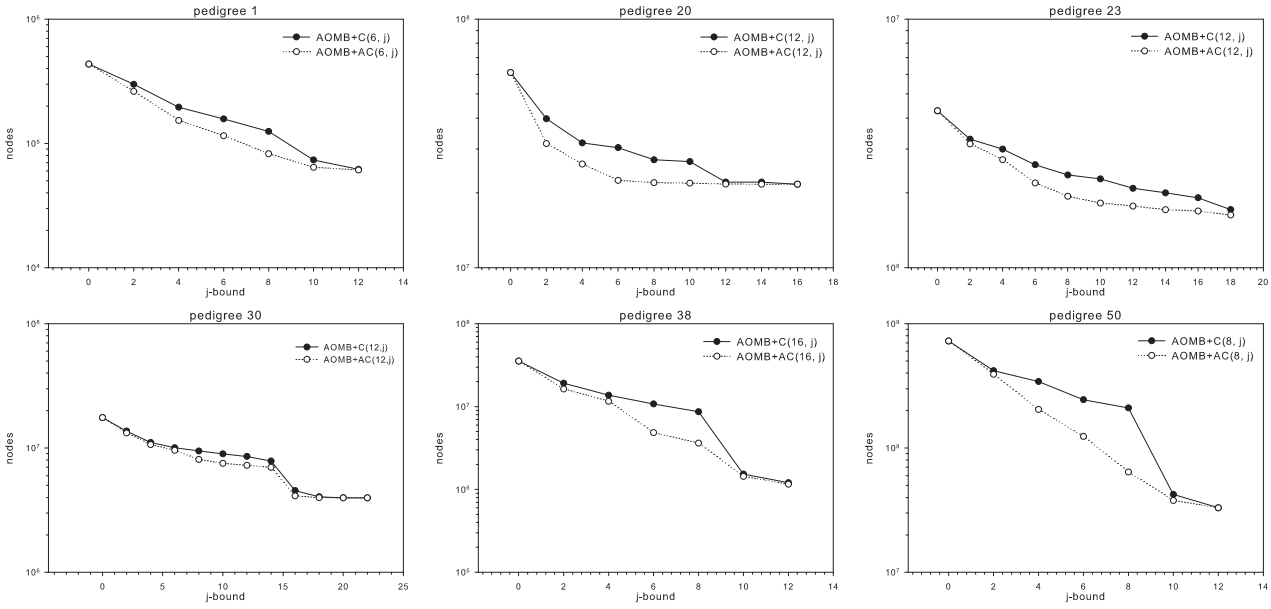



Figure 5: Detailed number of nodes visited comparing the naive vs. adaptive caching for genetic linkage analysis.

ings for the Branch-and-Bound algorithms.

## Progress from August 2006 to June 2007

In this section we summarize the progress made in the past year (since August 2006 until June 2007) on solving the maximum likelihood haplotype in genetic linkage analysis.

We first extend the AND/OR Branch-and-Bound algorithm by equipping it with a constraint propagation procedure that aims at exploiting in an efficient manner the determinism present in the belief network ($i.e.$, zero probability tuples in the CPTs).

Second, since depth-first AND/OR Branch-and-Bound algorithms were shown to be very effective when exploring such search spaces, especially when using caching and since best-first strategies are known to be superior to depth-first when memory is utilized, exploring the best-first control strategy is called for. The main contribution is in showing that a recent extension of AND/OR search algorithms from depth-first Branch-and-Bound to best-first is indeed very effective for computing the MPE in Bayesian networks.

We demonstrate empirically the superiority of the best-first search approach on several linkage analysis networks.

### AND/OR Branch-and-Bound with Constraint Propagation

In general, when the graphical model's functions express both hard constraints and general cost functions, it is beneficial to exploit the computational power of the constraints explicitly (Dechter & Larkin 2001; Larkin & Dechter 2003; Allen & Darwiche 2003; Dechter & Mateescu 2004). The constraints can be represented explicitly, as in Integer Linear Programming, or are hidden within the cost functions, such as the zero probability tuples in belief networks or the $\infty$ cost tuples in Weighted CSPs. In this section we introduce an extension of the AND/OR Branch-and-Bound algorithm that aims at handling in an efficient manner deterministic relationships that may occur in graphical models such as belief networks.

**Extracting the Determinism From the Belief Network** The approach we take for handling the determinism in belief networks is based on a key technique in the Boolean Satisfiability (SAT) literature, known as *unit resolution* over a logical knowledge base (KB) in the form of propositional clauses ($i.e.$, CNF formula). The CNF formula encodes the determinism in the network and is created based on the zero CPT entries, as follows.

**SAT Variables** Given a belief network $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$, the CNF is defined over the multi-valued variables $\{X_1, ..., X_n\}$. Its propositions are $L_{X_i,x_i}$, where $x_i \in D_i$. The proposition is true if $X_i$ is assigned value $x_i \in D_i$ and is false otherwise.

**SAT Clauses** The CNF is augmented with a collection of 2-CNFs for each variable $X_i$ in the network, called *at-most-one* clauses, that forbids the assignments of more than one value to a variable. Formally,

DEFINITION **9 (at-most-one clause)** *Given a variable $X_i \in \mathcal{X}$ with domain $D_i = \{x_{i_1}, ..., x_{i_d}\}$, its corresponding* at-most-one *clauses have the following form:*

$$\neg L_{X_i,x_{i_p}} \vee \neg L_{X_i,x_{i_q}}$$

*for every pair $(x_{i_p}, x_{i_q}) \in D_i \times D_i$, where $1 \leq p < q \leq d$.*

In addition, we will add to the CNF a set of *at-least-one* clauses to ensure that each variable in the network is assigned at least one value from its domain:

DEFINITION **10 (at-least-one clause)** *Given a variable $X_i \in \mathcal{X}$ with domain $D_i \in \{X_{i_1}, ..., X_{i_d}\}$, its corresponding* at-least-one *clause is of the following form:*

$$L_{X_i,x_{i_1}} \vee L_{X_i,x_{i_2}} ... \vee L_{X_i,x_{i_d}}$$

The remaining clauses are generated from the zero probability tuples in the network's CPTs.

DEFINITION **11 (no-good clauses)** *Given a conditional probability table $P(X_i|pa(X_i))$, each entry in the CPT having $P(x_i|x_{pa_i}) = 0$, where $pa(X_i) = \{Y_1, ..., Y_t\}$ are $X_i$'s parents and $x_{pa_i} = (y_1, ..., y_t)$ is their corresponding value assignment, can be translated to a* no-good *clause of the form:*

$$\neg L_{Y_1,y_1} \vee ... \vee \neg L_{Y_t,y_t} \vee \neg L_{X_i,x_i}$$

**Constraint Propagation via Unit Resolution** The input to unit resolution is a CNF formula $\varphi$, where each clause is a disjunction of literals. Each literal is either positive $(L = true)$ or negative $(\neg(L = true))$. A clause is satisfied whenever at least one of its literals is satisfied. Unit resolution is a linear time method for deriving logical implications of $\varphi$ based on setting the values of some variables, allowing one to efficiently detect variable assignments which are inconsistent with $\varphi$. The basic concept of unit resolution is that when all but one literal $L$ in a clause have been falsified, then literal $L$ must be satisfied in order to satisfy the clause.

Unit resolution is a very important part of any SAT solver, where a major portion of the solver's run time is spent doing it (Moskewicz *et al.* 2001). Our use of unit resolution within AND/OR Branch-and-Bound search is for detecting partial instantiations of the variables along the current path from the root that are guaranteed to have zero probabilities, and then skipping these instantiations.

**AOBB Search with Unit Resolution** In the AND/OR Branch-and-Bound algorithm unit resolution is implemented a `lookahead` function which is called upon the expansion of the current AND node $\langle X_i, x_i \rangle$.

When the algorithm expands the current AND node $\langle X_i, x_i \rangle$ in the forward step, the corresponding literal $L_{X_i,x_i}$ is set to *true*. The assertion is then propagated throughout the knowledge base via unit resolution. If a logical contradiction is encountered, then the AND node is marked as dead-end, and the search continues with the bottom-up cost revision step. During unit resolution, if a negative literal $\neg(L_{X_j,x_j} = true)$, corresponding to the uninstantiated variable $X_j$ of the current subproblem, is satisfied, then value $x_j$ can be safely removed from $X_j$'s domain, thus demonstrating the ability of the algorithm to prune future domains. Whenever the algorithm backtracks to the previous level, it also retracts any instantiation recorded by unit resolution.

## BEST-FIRST AND/OR SEARCH

In this section we direct our attention to a *best-first* rather than depth-first control strategy for traversing the context-minimal AND/OR graph and describe a best-first AND/OR search algorithm for solving the MPE task in belief networks. The algorithm uses similar amounts of memory as the depth-first AND/OR Branch-and-Bound with full caching and therefore the comparison is warranted.

**Algorithm 2:** AOBF

**Data**: A belief network $\mathcal{P} = \langle X, D, F \rangle$, pseudo-tree $T$, root $s$.
**Result**: Most Probable Explanation of $\mathcal{P}$.

1. Create explicit graph $G'_T$, consisting solely of the start node $s$. Set $v(s) = h(s)$.
2. **until** $s$ is labeled SOLVED, **do**:
   (a) Compute a *partial solution tree* by tracing down the *marked* arcs in $G'_T$ from $s$ and select any nonterminal tip node $n$.
   (b) Expand node $n$ and add any new successor node $n_i$ to $G'_T$. For each new node $n_i$ set $v(n_i) = h(n_i)$. Label SOLVED any of these successors that are terminal nodes.
   (c) Create a set $S$ containing node $n$.
   (d) **until** $S$ is empty, **do**:
      i. Remove from $S$ a node $m$ such that $m$ has no descendants in $G'_T$ still in $S$.
      ii. Revise the value $v(m)$ as follows:
         A. **if** $m$ is an AND node **then** $v(m) = \prod_{m_j \in succ(m)} v(m_j)$. If all the successor nodes are labeled SOLVED, then label node $m$ SOLVED.
         B. **if** $m$ is an OR node **then** $v(m) = max_{m_j \in succ(m)}(l(m, m_j) \cdot v(m_j))$ and mark the arc through which this maximum is achieved. If the marked successor is labeled SOLVED, then label $m$ SOLVED.
      iii. If $m$ has been marked SOLVED or if the revised value $v(m)$ is different than the previous one, then add to $S$ all those parents of $m$ such that $m$ is one of their successors through a marked arc.
3. **return** $v(s)$.

**Best-First Search** Best-first search is a search algorithm which optimizes breath-first search by expanding the node whose heuristic evaluation function is the best among all nodes encountered so far. Its main virtue is that it never expands nodes whose cost is beyond the optimal one, unlike depth-first search algorithms, and therefore is superior among memory intensive algorithms employing the same heuristic evaluation function (Dechter & Pearl 1985).

**Best-First AND/OR Graph Search** Our best-first AND/OR graph search algorithm, denoted by AOBF, that traverses the context-minimal AND/OR search graph is described in Algorithm 2. It specializes Nilsson's AO$^*$ algorithm (Nilsson 1980) to AND/OR spaces in graphical models, in particular to finding the MPE in belief networks.

The algorithm maintains a frontier of partial solution trees found so far, and interleaves forward expansion of the best partial solution tree with a cost revision step that updates estimated node values. First, a top-down, graph-growing operation (`step 2.a`) finds the best partial solution tree by tracing down through the marked arcs of the explicit AND/OR search graph $G'_T$. These previously computed marks indicate the current best partial solution tree from each node in $G'_T$. One of the nonterminal leaf nodes $n$ of this best partial solution tree is then expanded, and a static

heuristic estimate $h(n_i)$, overestimating $v(n_i)$, is assigned to its successors (`step 2.b`). The successors of an AND node $n = \langle X_j, x_j \rangle$ are $X_j$'s children in the pseudo-tree, while the successors of an OR node $n = X_j$ correspond to $X_j$'s domain values. Notice that when expanding an OR node, the algorithm does not generate AND children that are already present in the explicit search graph $G'_T$. All these identical AND nodes in $G'_T$ are easily recognized based on their contexts, so only pointers to the existing nodes are created.

The second operation in AOBF is a bottom-up, cost revision, arc marking, SOLVE-labeling procedure (`step 2.c`). Starting with the node just expanded $n$, the procedure revises its value $v(n)$ (using the newly computed values of its successors) and marks the outgoing arcs on the estimated best path to terminal nodes. This revised value is then propagated upwards in the graph. The revised cost $v(n)$ is an updated estimate of the most probable explanation probability of the subproblem rooted at $n$. If we assume the monotone restriction on $h$, the algorithm considers only those ancestors that root best partial solution subtrees containing descendants with revised values. The most probable explanation value of the initial problem is obtained when the root node $s$ is solved.

**AOBB versus AOBF Search** We describe next the main differences between AOBF and AOBB search.

1 AOBF with the same heuristic function as AOBB is likely to expand the smallest number of nodes (Dechter & Pearl 1985), but empirically this depends on how quickly AOBB will find an optimal solution.

2 AOBB is able to improve its heuristic function dynamically during search (Marinescu & Dechter 2005) based on the explicated portion of the search space, while AOBF may not because it uses only the static function $h(n)$, which can be pre-computed or generated during search.

3 AOBB can use far less memory avoiding dead-caches for example (e.g., when the search graph is a tree), while AOBF has to keep the explicated search graph in memory prior to termination.

All the above points show that the relative merit of best-first vs depth-first over context-minimal AND/OR search spaces cannot be determined by the theory in (Dechter & Pearl 1985) and empirical evaluation is essential.

## Experiments

We consider a class of best-first AND/OR search algorithms guided by the static mini-bucket heuristics and denoted by AOBF+SMB($i$) respectively. We compare it against the depth-first AND/OR Branch-and-Bound algorithms with static mini-bucket heuristics and full caching and denoted by AOBB+SMB($i$). The parameter $i$ represents the mini-bucket $i$-bound and controls the accuracy of the heuristic. All algorithms traverse the context-minimal AND/OR search graph and are restricted to a static variable ordering determined by the pseudo-tree.

**Effect of Unit Resolution** Table 2 shows the results obtained for 12 linkage analysis networks: {1, 18, 20, 23, 25, 30, 33, 37, 38, 39, 42, 50} from *http://bio.cs.technion.ac.il/superlink/*. We considered the AND/OR Branch-and-Bound algorithms with static mini-bucket heuristics, no caching and with either unit resolution, denoted by AOBB+SAT+SMB($i$), or without constraint propagation, denoted by AOBB+SMB($i$), respectively. We observe that unit resolution is not cost effective in this case.

**AOBB versus AOBF** Table 3 displays the results obtained for the same set of 12 linkage analysis networks. For comparison, we include results obtained with SUPERLINK 1.6. SUPERLINK (Fishelson & Geiger 2002; Fishelson, Dovgolevsky, & Geiger 2005) is currently one the most efficient solvers 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.

In addition, we also ran SAMIAM version 2.3.2 and RC-LINK solvers. SAMIAM[2] is a public implementation of Recursive Conditioning (Darwiche 2001) which can also be viewed as an AND/OR graph search algorithm for solving the MPE task. RC-LINK[3] is also a Recursive Conditioning based algorithm, however it is specialized for the linkage analysis domain and it can only compute the probability of evidence of the belief network that corresponds to the input linkage network.

In all our experiments, AOBB+SMB($i$), AOBF+SMB($i$), SAMIAM and SUPERLINK were run on the belief network output by SUPERLINK. RC-LINK used its own simplified belief network (to which we did not have access).

When comparing the AND/OR search algorithms, we observe that AOBF+SMB($i$) is the best performing algorithm. For instance, on the p42 linkage instance, AOBF+SMB(14) is 18 times faster than AOBB+SMB(14) and explores a search space 240 times smaller. On some instances (e.g., p1, p23, p30) the best-first search algorithm AOBF+SMB($i$) is several orders of magnitude faster than SUPERLINK. The performance of SAMIAM was very poor on this dataset and it was able to solve only 2 instances.

Table 4 displays the resuts obtained for the remaining 10 linkage networks from the dataset available at *http://bio.cs.technion.ac.il/superlink/*, namely {7, 9, 13, 19, 31, 34, 40, 41, 44, 51}. These networks cannot be solved by either AOBB+SMB($i$) or AOBF+SMB($i$), within a 3 hour time limit for all reported $i$-bounds.

On the other hand, RC-LINK appears to be the best performing algorithm on this dataset. This can be explained by its very powerful reduction methods that can produce a far simpler belief network as compared to the one output by SUPERLINK.

[2] Available at http://reasoning.cs.ucla.edu/samiam. We used the `batchtool 1.5` provided with the package.

[3] Available at http://reasoning.cs.ucla.edu/rc_link

| ped (n,d,w*,h) | Superlink v. 1.6 | SamIam v. 2.3.2 | AOBB+SMB(i) AOBB+SAT+SMB(i) i=6 | | AOBB+SMB(i) AOBB+SAT+SMB(i) i=8 | | AOBB+SMB(i) AOBB+SAT+SMB(i) i=10 | | AOBB+SMB(i) AOBB+SAT+SMB(i) i=12 | | AOBB+SMB(i) AOBB+SAT+SMB(i) i=14 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | time | nodes | time | nodes | time | nodes | time | nodes | time | nodes |
| **ped1** | 54.73 | 5.44 | 24.30 | 416,326 | 13.17 | 206,439 | **1.58** | 24,361 | 1.84 | 25,674 | 1.89 | 15,156 |
| (299,5,15,61) | | | 24.72 | 414,239 | 12.97 | 205,887 | 1.59 | 24,361 | 1.86 | 25,674 | 1.89 | 15,156 |
| **ped38** | **28.36** | out | - | - | 8120.58 | 85,367,022 | - | - | 3040.60 | 35,394,461 | | |
| (582,5,17,59) | | | - | - | 7663.89 | 83,808,576 | - | - | 3094.33 | 35,394,277 | | |
| **ped50** | - | out | - | - | - | - | 476.77 | 5,566,578 | **104.00** | 748,792 | | |
| (479,5,18,58) | | | - | - | - | - | 497.30 | 5,566,344 | 107.11 | 748,792 | | |
| | | | i=10 | | i=12 | | i=14 | | i=16 | | i=18 | |
| **ped23** | 9146.19 | out | 102.48 | 1,375,196 | 87.34 | 1,149,195 | 13.08 | 145,330 | 3.22 | 21,351 | **3.13** | 11,132 |
| (310,5,23,37) | | | 105.83 | 1,375,196 | 88.50 | 1,149,195 | 13.53 | 145,330 | 3.25 | 21,351 | 3.16 | 11,132 |
| **ped37** | **64.17** | out | 273.39 | 3,191,218 | 1682.09 | 25,729,009 | 1096.79 | 15,598,863 | 128.16 | 953,061 | | |
| (1032,5,21,61) | | | 282.83 | 3,189,847 | 1674.54 | 25,280,466 | 1066.79 | 15,372,724 | 131.56 | 953,061 | | |
| | | | i=12 | | i=14 | | i=16 | | i=18 | | i=20 | |
| **ped18** | 139.06 | 157.05 | - | - | 2177.81 | 28,651,103 | 270.96 | 2,555,078 | 100.61 | 682,175 | **20.27** | 7,689 |
| (1184,5,21,119) | | | - | - | 2199.44 | 28,651,103 | 285.03 | 2,555,078 | 103.89 | 682,175 | 20.41 | 7,689 |
| **ped20** | **14.72** | out | - | - | - | - | 38.75 | 311,385 | 96.02 | 555,872 | | |
| (388,5,23,42) | | | - | - | - | - | 40.49 | 311,385 | 100.20 | 555,872 | | |
| **ped25** | - | out | - | - | - | - | - | - | 8415.18 | 45,825,494 | **1894.17** | 11,709,153 |
| (994,5,29,53) | | | - | - | - | - | - | - | 7514.83 | 45,824,181 | 1972.51 | 11,710,927 |
| **ped30** | 13095.83 | out | 5563.22 | 63,068,960 | 1397.14 | 15,336,772 | 1811.34 | 20,275,620 | 550.57 | 5,535,261 | **82.25** | 588,558 |
| (1016,5,25,51) | | | 5728.35 | 63,068,960 | 1409.02 | 15,336,548 | 1857.48 | 20,275,620 | 577.82 | 5,535,261 | 84.38 | 588,558 |
| **ped33** | - | out | 2335.28 | 32,444,818 | 806.12 | 11,403,812 | **62.91** | 807,071 | 67.92 | 701,030 | 76.47 | 320,279 |
| (581,5,26,48) | | | 2378.66 | 32,444,818 | 820.73 | 11,403,812 | 63.99 | 807,071 | 69.39 | 701,030 | 77.20 | 320,279 |
| **ped39** | 322.14 | out | - | - | - | - | 4041.56 | 52,804,044 | 386.13 | 2,171,470 | **141.23** | 407,280 |
| (1272,5,23,94) | | | - | - | - | - | 4242.59 | 52,804,044 | 405.08 | 2,171,470 | 145.03 | 407,280 |
| **ped42** | **561.31** | out | - | - | - | - | - | - | | | | |
| (448,5,25,76) | | | - | - | - | - | - | - | | | | |

Table 2: CPU time in seconds and number of nodes visited for solving genetic linkage analysis networks. Time limit 3 hours.

| ped | n | w* | | RC-LINK | SamIam | Superlink | AOBB+SMB(i) | | | | | AOBF+SMB(i) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | d | h | | | v. 2.3.2 | v. 1.6 | i=6 | i=8 | i=10 | i=12 | i=14 | i=6 | i=8 | i=10 | i=12 | i=14 |
| 1 | 299 | 15 | t | 24.50 | 5.44 | 54.73 | 4.19 | 2.17 | 0.39 | 0.65 | 1.36 | 1.30 | 2.17 | **0.26** | 0.87 | 1.54 |
| | | 61 | # | | | | 69,751 | 33,908 | 4,576 | 6,306 | 4,494 | 7,314 | 13,784 | 1,177 | 4,016 | 3,119 |
| 38 | 582 | 17 | t | 425.40 | out | **28.36** | 5946.44 | 1554.65 | 2046.95 | 272.69 | n/a | out | 134.41 | 216.94 | 103.17 | n/a |
| | | 59 | # | | | | 34,828,046 | 8,986,648 | 11,868,672 | 1,412,976 | | | 348,723 | 583,401 | 242,429 | |
| 50 | 479 | 18 | t | 26411.60 | out | - | 4140.29 | 2493.75 | 66.66 | 52.11 | n/a | 78.53 | 36.03 | **12.75** | 38.52 | n/a |
| | | 58 | # | | | | 28,201,843 | 15,729,294 | 403,234 | 110,302 | | 204,886 | 104,289 | 25,507 | 5,766 | |
| | | | | | | | i=10 | i=12 | i=14 | i=16 | i=18 | i=10 | i=12 | i=14 | i=16 | i=18 |
| 23 | 310 | 23 | t | 3.20 | out | 9146.19 | 53.70 | 49.33 | 8.77 | **2.73** | 3.04 | 35.49 | 29.29 | 10.59 | 3.59 | 3.48 |
| | | 37 | # | | | | 486,991 | 437,688 | 85,721 | 14,019 | 7,089 | 185,761 | 150,214 | 52,710 | 11,414 | 5,790 |
| 37 | 1032 | 21 | t | 66.20 | out | 64.17 | 39.16 | 488.34 | 301.78 | 67.83 | n/a | **29.16** | 38.41 | 95.27 | 62.97 | n/a |
| | | 61 | # | | | | 222,747 | 4,925,737 | 2,798,044 | 82,239 | | 72,868 | 102,011 | 223,398 | 12,296 | |
| | | | | | | | i=12 | i=14 | i=16 | i=18 | i=20 | i=12 | i=14 | i=16 | i=18 | i=20 |
| 18 | 1184 | 21 | t | **5.40** | 157.05 | 139.06 | - | 406.88 | 52.91 | 23.83 | 20.60 | out | 127.41 | 42.19 | **19.85** | 19.91 |
| | | 119 | # | | | | | 3,567,729 | 397,934 | 118,869 | 2,972 | | 542,156 | 171,039 | 53,961 | 2,027 |
| 20 | 388 | 23 | t | 34.10 | out | **14.72** | 7243.43 | 5560.63 | 37.28 | 95.13 | n/a | out | out | 33.33 | 121.91 | n/a |
| | | 42 | # | | | | 63,530,037 | 46,858,127 | 279,804 | 554,623 | | | | 144,212 | 466,817 | |
| 30 | 1016 | 25 | t | **5.90** | out | 13095.83 | 1440.26 | 597.88 | 1023.90 | 151.96 | 43.83 | 186.77 | 58.38 | 85.53 | 49.38 | **33.03** |
| | | 51 | # | | | | 11,694,534 | 5,580,555 | 10,458,174 | 1,179,236 | 146,896 | 692,870 | 253,465 | 350,497 | 179,790 | 37,705 |
| 39 | 1272 | 23 | t | **9.90** | out | 322.14 | - | - | 968.03 | 61.20 | 93.19 | out | out | 68.52 | **41.69** | 87.63 |
| | | 94 | # | | | | | | 7,880,928 | 313,496 | 83,714 | | | 218,925 | 79,356 | 14,479 |
| 42 | 448 | 25 | t | **36.20** | out | 561.31 | - | - | 2364.67 | n/a | n/a | out | out | **133.19** | n/a | n/a |
| | | 76 | # | | | | | | 22,595,247 | | | | | 93,831 | | |
| 25 | 994 | 29 | t | **26.70** | out | - | - | - | - | 2041.64 | 693.74 | out | out | out | out | **198.49** |
| | | 53 | # | | | | | | | 6,117,320 | 1,925,152 | | | | | 468,723 |
| 33 | 581 | 26 | t | **3.90** | out | - | 886.05 | 370.41 | 26.31 | 33.11 | 54.89 | out | 194.78 | **24.16** | 32.55 | 58.52 |
| | | 48 | # | | | | 8,426,659 | 4,032,864 | 229,856 | 219,047 | 83,360 | | 975,617 | 102,888 | 101,862 | 57,593 |

Table 3: CPU time in seconds and number of nodes visited for genetic linkage analysis. Time limit 3 hours.

| ped | n<br>d | w*<br>h | | RC-LINK | SamIam<br>v. 2.3.2 | Superlink<br>v. 1.6 | AOBB+SMB(i)<br>i=12 | <br>i=14 | <br>i=16 | <br>i=18 | <br>i=20 | AOBF+SMB(i)<br>i=12 | <br>i=14 | <br>i=16 | <br>i=18 | <br>i=20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 1068<br>4 | 39<br>147 | t<br># | **22.90** | out | - | | | | | | | | | | |
| 9 | 1118<br>7 | 31<br>107 | t<br># | **168.60** | out | out | | | | | | | | | | |
| 13 | 1077<br>3 | 35<br>174 | t<br># | **98.70** | out | - | | | | | | | | | | |
| 19 | 793<br>5 | 28<br>155 | t<br># | **19.20** | out | out | | | | | | | | | | |
| 31 | 1183<br>5 | 34<br>108 | t<br># | **144.90** | out | - | | | | | | | | | | |
| 34 | 1160<br>5 | 44<br>118 | t<br># | **31.70** | out | out | | | | | | | | | | |
| 40 | 1030<br>7 | 36<br>153 | t<br># | **57.60** | out | - | | | | | | | | | | |
| 41 | 1062<br>5 | 41<br>119 | t<br># | **13.40** | out | - | | | | | | | | | | |
| 44 | 811<br>4 | 25<br>90 | t<br># | **121.40** | out | - | | | | | | | | | | |
| 51 | 1152<br>5 | 47<br>106 | t<br># | **556.50** | out | - | | | | | | | | | | |

Table 4: Results obtained for the remaining 10 linkage networks. These networks could not be solved by neither AOBB nor AOBF, within a 3 hour time limit. Similarly, SUPERLINK runs out of memory or time.

## 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


Allen, D., and Darwiche, A. 2003. New advances in inference using recursive conditioning. *In Uncertainty in Artificial Intelligence (UAI-2003)* 2–10.

Bacchus, F.; Dalmao, S.; and Pittasi, T. 2003. Value elimination: Bayesian inference via backtracking search. *In Uncertainty in Artificial Intelligence (UAI-2003)* 20–28.

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

Dechter, R., and Larkin, D. 2001. Hybrid processing of beliefs and constraints. *In Uncertainty in Artificial Intelligence (UAI-2001)* 112–119.

Dechter, R., and Mateescu, R. 2004. Mixtures of deterministic-probabilistic networks. *In Uncertainty in Artificial Intelligence (UAI-2004)*.

Dechter, R., and Mateescu, R. 2006. And/or search spaces for graphical models. *Artificial Intelligence*.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of a*. *In Journal of ACM* 32(3):505–536.

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

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113:41–85.

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 constraint satisfaction problems. *In International Joint Conference on Artificial Intelligence (IJCAI-1985)* 1076–1078.

Jegou, P., and Terrioux, C. 2004. Decomposition and good recording for solving max-csps. *In European Conference on Artificial Intelligence (ECAI 2004)* 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*.

Larkin, D., and Dechter, R. 2003. Bayesian inference in the presence of determinism. *In Artificial Intelligence and Statistics (AISTAT-2003)*.

Larrosa, J.; Meseguer, P.; and Sanchez, M. 2002. Pseudo-tree search with soft constraints. *In European Conference on Artificial Intelligence (ECAI-2002)* 131–135.

Marinescu, R., and Dechter, R. 2005. And/or branch-and-bound for graphical models. *In International Joint Conference on Artificial Intelligence (IJCAI-2005)* 224–229.

Mateescu, R., and Dechter, R. 2005. And/or cutset conditioning. *In International Joint Conference on Artificial Intelligence (IJCAI-2005)* 230–235.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. *In Design Automation Conference (DAC-2001)*.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence.* Tioga.

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

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