Best-First AND/OR Search for Graphical Models

Radu Marinescu and **Rina Dechter**
School of Information and Computer Science
University of California, Irvine, CA 92627
{radum,dechter}@ics.uci.edu

Abstract

The paper presents and evaluates the power of *best-first search* over AND/OR search spaces in graphical models. The main virtue of the AND/OR representation is its sensitivity to the structure of the graphical model, which can translate into significant time savings. Indeed, in recent years depth-first AND/OR Branch-and-Bound algorithms were shown to be very effective when exploring such search spaces, especially when using caching. 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. In this paper we introduce two classes of best-first AND/OR search algorithms: those that explore a context-minimal AND/OR search graph and use static variable orderings, and those that use dynamic variable orderings but explore an AND/OR search tree. The superiority of the best-first search approach is demonstrated empirically on various real-world benchmarks.

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 problems 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.

The AND/OR search space for graphical models (Dechter & Mateescu 2006) is a framework for search that is sensitive to the independencies in the model, often resulting in reduced search spaces. The impact of the AND/OR search to optimization in graphical models was explored in recent years focusing exclusively on depth-first search.

The AND/OR Branch-and-Bound first introduced by (Marinescu & Dechter 2005) traverses the AND/OR search tree in a depth-first manner. The memory intensive Branch-and-Bound algorithm (Marinescu & Dechter 2006c) explores an AND/OR search graph, rather than a tree, by caching previously computed results and retrieving them when the same subproblems are encountered again. These algorithms were initially restricted to a static variable ordering. More recently, (Marinescu & Dechter 2006b) showed how dynamic variable selection heuristics influence Branch-and-Bound search over AND/OR trees. The depth-first AND/OR search algorithms were shown to outperform dramatically state-of-the-art Branch-and-Bound algorithms searching the traditional OR space.

In this paper we focus on best-first search algorithms. We present a new AND/OR search algorithm that explores a context-minimal AND/OR search graph in a *best-first* rather than depth-first manner. Since variable selection can have a dramatic impact on search performance, we also introduce a best-first AND/OR search algorithm that explores the AND/OR search tree, rather than the graph, and combines the static AND/OR decomposition principle with dynamic variable selection heuristics. Under conditions of admissibility and monotonicity of the heuristic function, best-first search is known to expand the minimal number of nodes, at the expense of using additional memory (Dechter & Pearl 1985). In practice, these savings in number of nodes may often translate into time savings as well.

We focus the empirical evaluation on three common optimization problems: solving Weighted CSPs (de Givry *et al.* 2005), finding the Most Probable Explanation in belief networks (Pearl 1988), and 0/1 Integer Linear Programming (Nemhauser & Wolsey 1988). Our results show conclusively that the best-first AND/OR search approach outperforms significantly the depth-first AND/OR Branch-and-Bound search algorithms on various benchmarks.

Background

Constraint Optimization Problems

A finite *Constraint Optimization Problem* (COP) is a triple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$, where $\mathcal{X} = \{X_1, ..., X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, ..., D_n\}$ is a set of finite domains and $\mathcal{F} = \{f_1, ..., f_m\}$ is a set of cost functions. Cost functions can be either *soft* or *hard* (constraints). Without loss of generality we assume that hard constraints are represented as (bi-valued) cost functions. Allowed and forbidden tuples have cost 0 and ∞, respectively. The scope of function f_i, denoted $scope(f_i) \subseteq \mathcal{X}$, is the set of arguments of f_i. The goal is to find a complete value assignment to the vari-

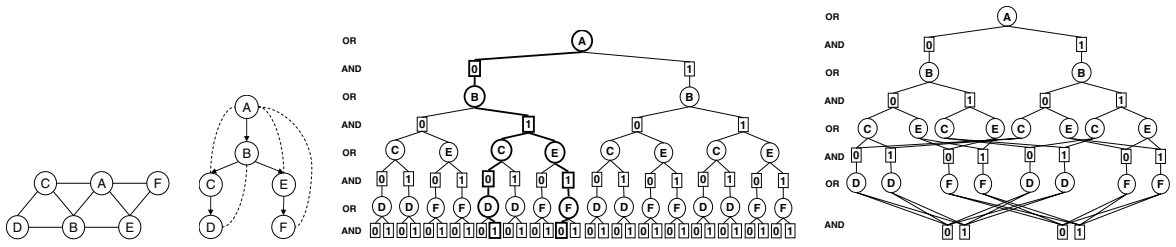

Figure 1: AND/OR Search Spaces for Graphical Models

ables that minimizes the global cost function, namely to find $x = \arg\min_{\mathcal{X}} \sum_{i=1}^{m} f_i$.

Given a COP instance, its *primal graph* G associates each variable with a node and connects any two nodes whose variables appear in the scope of the same function.

AND/OR Search Spaces for Graphical Models

The usual way to do search is to instantiate variables, following a static/dynamic variable ordering. In the simplest case, this process defines an OR search tree, whose nodes represent partial assignments. This search space does not capture the structure of the underlying graphical model. However, to remedy this problem, AND/OR search spaces for graphical models were recently introduced by (Dechter & Mateescu 2006). They are defined using a backbone *pseudo-tree* (Freuder & Quinn 1985).

DEFINITION **1 (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.*

AND/OR Search Trees Given a COP instance $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, its primal graph G and a pseudo-tree T of G, the associated AND/OR search tree, denoted 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 root of the AND/OR search tree is an OR node, labeled with the root of the pseudo-tree 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. Semantically, the OR states represent alternative solutions, 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 tree* 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_{S_T} then exactly one of its children is in Sol_{S_T}.

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

Arc Labels and Node Values The arcs from OR nodes X_i to AND nodes $\langle X_i, x_i \rangle$ in the AND/OR search tree S_T are annotated by *labels* derived from the cost functions in $\mathcal{F}$.

DEFINITION **2 (label)** *The* label $l(X_i, \langle X_i, x_i \rangle)$ *of the arc from the OR node X_i to the AND node $\langle X_i, x_i \rangle$ is the sum of all the cost functions whose scope includes X_i and is fully assigned along $path(X_i, x_i)$, evaluated at the values along the path.*

Given a labeled AND/OR search tree, each node can be associated with a *value* (Dechter & Mateescu 2006).

DEFINITION **3 (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 $v(n) = 0$; (ii) if $n = \langle X_i, x_i \rangle$ is an internal AND node then $v(n) = \sum_{n' \in succ(n)} v(n')$; (iii) if $n = X_i$ is an internal OR node then $v(n) = min_{n' \in succ(n)}(l(n, n') + v(n'))$, where $succ(n)$ are the children of n in S_T.*

It is easy to see that the value $v(n)$ of a node in the AND/OR search tree S_T is the minimal cost solution to the subproblem rooted at n, subject to the current variable instantiation along the path from the root to n. If n is the root of S_T, then $v(n)$ is the minimal cost solution to the initial problem (Marinescu & Dechter 2005).

AND/OR Search Graphs The AND/OR search tree may contain nodes that root identical subtrees (in particular, subproblems with identical optimal solutions) which can be *unified*. When unifiable nodes are merged, the search tree becomes a graph and its size becomes smaller. Some unifiable nodes can be identified based on their *contexts*.

DEFINITION **4 (context)** *Given a COP instance 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 any two nodes having the same context represent the same subproblem. Therefore, we can solve P_{X_i}, the subproblem rooted at X_i, once and use its

optimal solution whenever the same subproblem is encountered again.

The *context-minimal* AND/OR search graph based on pseudo-tree T, denoted G_T, is obtained by merging all the AND nodes that have the same context. It can be shown (Dechter & Mateescu 2006) that the size of the largest context is bounded by the induced width w^* of the primal 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,

THEOREM **1 (complexity)** *The complexity of any search algorithm traversing a context-minimal AND/OR search graph is time and space $O(exp(w^*))$, where w^* is the induced width of the underlying pseudo-tree.*

Example 2 *Consider the context-minimal AND/OR search graph in Figure 1(d) of the pseudo-tree from Figure 1(b). Its size is far smaller than that of the AND/OR tree from Figure 1(c) (16 nodes vs. 36 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}.*

Best-First AND/OR Search

In recent years, depth-first AND/OR Branch-and-Bound algorithms were shown to be very effective, especially when using extensive caching (Marinescu & Dechter 2005; 2006c). Since best-first search is known to be superior among memory intensive search algorithms (Dechter & Pearl 1985), the comparison with the best-first approach that exploits similar amounts of memory is warranted. In this section we introduce two new classes of best-first AND/OR search algorithms: one that explores a context-minimal AND/OR search graph and is restricted to a static variable ordering, and one that uses dynamic variable orderings but traverses an AND/OR search tree, rather than a graph.

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 1. It specializes Nillson's AO* algorithm (Nillson 1980) to AND/OR spaces in graphical models. The algorithm 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 $h(n_i)$, underestimating $v(n_i)$, 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 generates 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.

Algorithm 1: AOBF

Data: A COP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, pseudo-tree T, root s.
Result: Minimal cost solution to $\mathcal{P}$.

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** $v(m) = \sum_{m_j \in succ(m)} v(m_j)$. If all the successor nodes are labeled SOLVED, then label node m SOLVED.
         B. **if** m is an OR node **then** $v(m) = min_{m_j \in succ(m)}(l(m, m_j) + v(m_j))$ and mark the arc through which this minimum 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** $v(s)$.

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 cost of an optimal solution to 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 optimal cost solution to the initial problem is obtained when the root node s is solved.

Dynamic Variable Orderings

It is well known that variable selection may influence dramatically search performance. Recent work by (Marinescu & Dechter 2006b) showed how several dynamic variable orderings affect depth-first Branch-and-Bound search over AND/OR trees. One version, called AND/OR Branch-and-Bound with Partial Variable Ordering (AOBB+PVO) that orders dynamically the variables forming chains in the pseudo-tree was shown to outperform significantly static AND/OR search as well as state-of-the-art OR Branch-and-Bound solvers for general COPs. Next, we extend the idea of partial variable ordering to best-first search on AND/OR trees.

Note that AOBF is restricted to a static variable ordering that corresponds to the pseudo-tree arrangement. The mechanism of identifying unifiable AND nodes based solely on

spot5	n	w^*		toolbar3	AOEDAC	AOBBMB(i)					AOBFMB(i)				
	c	h			DVO	i=4	i=6	i=8	i=10	i=12	i=4	i=6	i=8	i=10	i=12
29	83	14	time	4.56	0.81	5.53	4.80	0.56	3.64	21.67	6.42	2.23	**0.47**	3.59	21.77
	476	42	nodes	218,846	8,698	48,995	29,702	2,267	1,165	110	36,396	12,801	757	323	96
						i=2	i=4	i=6	i=8	i=10	i=2	i=4	i=6	i=8	i=10
54	68	11	time	0.31	**0.06**	546.89	18.42	0.23	0.16	0.69	0.69	0.41	0.11	0.16	0.69
	283	33	nodes	21,939	688	5,094,051	198,712	2,477	591	120	3,906	2,714	631	312	68
						i=6	i=8	i=10	i=12	i=14	i=6	i=8	i=10	i=12	i=14
404	100	19	time	151.11	12.09	51.88	2.55	**0.55**	1.16	3.98	1.20	1.02	0.62	1.22	4.00
	710	42	nodes	6,215,135	88,079	529,002	23,565	1,704	598	232	6,399	5,140	1,303	576	184
						i=6	i=8	i=10	i=12	i=14	i=6	i=8	i=10	i=12	i=14
408b	201	24	time	-	-	-	7507.10	515.94	75.08	47.03	52.53	44.99	25.20	**16.97**	38.53
	1847	59	nodes				54,826,929	3,114,294	408,619	61,986	175,366	145,901	98,616	39,238	14,768
						i=2	i=4	i=6	i=8	i=10	i=2	i=4	i=6	i=8	i=10
503	144	9	time	-	10005.00	-	-	189.39	291.72	**0.42**	10.25	5.28	1.56	1.59	**0.42**
	639	39	nodes		44,495,545			2,442,998	4,050,474	256	22,967	16,114	9,929	9,186	144
						i=6	i=8	i=10	i=12	i=14	i=6	i=8	i=10	i=12	i=14
505b	241	16	time	-	-	-	-	-	1180.48	367.93	42.73	**29.25**	31.20	54.09	373.72
	1721	98	nodes						8,905,473	16,020	144,723	111,223	108,256	31,692	5,758

Table 1: CPU time in seconds and nodes visited to prove optimality for SPOT5 benchmarks. Time limit 3 hours.

their contexts is hard to extend when variables are instantiated in a different order than that dictated by the pseudo-tree.

Best-first AND/OR search with Partial Variable Ordering (AOBF+PVO) traverses an AND/OR search tree in a best-first manner and combines the static graph-based problem decomposition given by a pseudo-tree with a dynamic semantic variable selection heuristic. We illustrate the idea with an example. Consider the pseudo-tree from Figure 1(b) inducing the following variable group ordering: {A,B}, {C,D}, {E,F}; which dictates that variables {A,B} should be considered before {C,D} and {E,F}. Variables in each group (or chain) can be dynamically ordered based on a second, independent heuristic. Notice that after variables {A,B} are instantiated, the problem decomposes into two independent components (represented by variables {C,D} and {E,F}, resp.) that can be solved separately.

Experiments

We evaluate the performance of the two classes of best-first AND/OR search algorithms on three common optimization problems: solving Weighted CSPs (de Givry *et al.* 2005), finding the Most Probable Explanation (MPE) in belief networks (Pearl 1988) and solving 0/1 Integer Linear Programs (Nemhauser & Wolsey 1988). All experiments were run on a 2.4GHz Pentium IV with 2GB of RAM.

We report the average CPU time (in seconds) and number of nodes visited, required for proving optimality of the solution. We also record the number of variables (n), the number of constraints (c), the depth of the pseudo-trees (h) and the induced width of the graphs (w^*) obtained for the test instances. The pseudo-trees were generated using the min-fill heuristic, as described in (Marinescu & Dechter 2005). The best performance points are highlighted.

Weighted CSPs

For this domain we experimented with real-world scheduling and circuit diagnosis benchmarks. We consider the best-first AND/OR search algorithm guided by pre-compiled mini-bucket heuristics (Marinescu & Dechter 2005) and denoted by AOBFMB(i). We compare it against the depth-first AND/OR Branch-and-Bound algorithm with static mini-bucket heuristics and full caching introduced by (Marinescu & Dechter 2006c) and denoted by AOBBMB(i). The parameter i represents the mini-bucket i-bound and controls the accuracy of the heuristic. Both algorithms traverse the context-minimal AND/OR search graph restricted to a static variable ordering determined by the pseudo-tree.

We also report results obtained with the OR Branch-and-Bound maintaining Existential Directional Arc-Consistency (EDAC) developed in (de Givry *et al.* 2005) and denoted by toolbar3, and the AND/OR Branch-and-Bound with EDAC and full dynamic variable ordering (AOEDAC+DVO) from (Marinescu & Dechter 2006b). These algorithms instantiate variables dynamically, using the *min-dom/deg* heuristic which selects the variable with the smallest ratio of the domain size divided by the future degree.

Earth Observing Satellites The problem of scheduling an Earth observing satellite is to select from a set of candidate photographs, the best subset such that a set of imperative constraints are satisfied and the total importance of the selected photographs is maximized. We experimented with problem instances from the SPOT5 benchmark (Bensana, Lemaitre, & Verfaillie 1999) which can be formulated as WCSPs with binary and ternary constraints and domain sizes of 2 and 4 (instances 408b and 505b contain only binary constraints).

Table 1 shows the results for experiments with 6 scheduling problems. The columns are indexed by the mini-bucket i-bound. When comparing the best-first against the depth-first AND/OR search algorithms with static mini-bucket heuristics we observe that, for relatively small i-bounds, AOBFMB(i) improves significantly (up to several orders of magnitude) in terms of both CPU time and number of nodes visited. For example, on 505b, one of the hardest instances, AOBFMB(8) proves optimality in less than 30 seconds, whereas AOBBMB(8) exceeds the 3 hour time limit. This observation verifies the theory because best-first search is likely to expand the smallest number of nodes at the search frontier, especially when having relatively weak heuristic estimates. As the mini-bucket i-bound increases and the heuristics become strong enough to cut the search space substantially the difference between Branch-and-Bound and best-first search decreases, because Branch-and-Bound finds almost optimal solutions fast, and therefore will not explore

iscas89	n	w^*		toolbar3	AOBB+EDAC	AOBBMB(i)					AOBFMB(i)				
	c	h			DVO	i=8	i=10	i=12	i=14	i=16	i=8	i=10	i=12	i=14	i=16
c432	432	27	time	-	-	422.08	40.91	0.89	0.89	0.64	39.33	0.52	**0.31**	0.38	0.67
	432	45	nodes			2,945,230	337,574	6,254	6,010	914	196,892	2,154	1,007	847	445
c880	881	27	time	-	-	100.66	91.66	31.06	59.35	14.78	1.36	0.91	**0.81**	1.19	1.44
	883	67	nodes			516,056	446,893	169,138	316,124	78,268	4,454	2,792	2,231	2,862	1,589
s935	441	66	time	-	-	1285.07	143.53	-	22.28	4.80	6.16	1.22	**1.19**	1.22	2.42
	464	101	nodes			6,623,608	763,933		128,372	15,010	25,493	4,087	3,319	2,216	883
s1196	562	54	time	-	-	3347.38	503.30	2299.72	734.66	149.81	22.67	**2.89**	13.02	7.27	3.56
	564	97	nodes			13,554,137	2,425,152	11,488,366	3,524,780	793,417	72,075	9,336	40,210	21,989	2,090
s1238	541	59	time	-	-	1897.37	1682.99	281.05	248.27	12.64	34.09	29.41	12.31	6.64	**4.63**
	543	94	nodes			8,386,634	7,431,223	1,350,933	1,220,658	59,635	137,960	111,205	53,095	26,101	7,142
s1494	661	48	time	-	-	364.80	5.64	27.64	6.92	9.02	1.44	**0.59**	0.95	1.50	3.81
	661	69	nodes			953,945	17,279	80,895	23,131	20,004	5,694	1,472	2,311	1,476	985

Table 2: CPU time in seconds and nodes visited to prove optimality for ISCAS'89 benchmarks. Time limit 1 hour.

ped	n	w^*		Superlink 1.5	AOBBMB(i)					AOBFMB(i)				
	c	h			i=6	i=8	i=10	i=12	i=14	i=6	i=8	i=10	i=12	i=14
1	299	15	time	131.30	4.19	2.17	0.39	0.65	1.36	1.30	2.17	**0.26**	0.87	1.54
	340	61	nodes		69,751	33,908	4,576	6,306	4,494	7,314	13,784	1,177	4,016	3,119
					i=10	i=12	i=14	i=16	i=18	i=10	i=12	i=14	i=16	i=18
23	310	23	time	6,809	53.70	49.33	**8.77**	2.73	3.04	35.49	29.29	10.59	3.59	3.48
	410	37	nodes		486,991	437,688	85,721	14,019	7,089	185,761	150,214	52,710	11,414	5,790
					i=12	i=14	i=16	i=18	i=20	i=12	i=14	i=16	i=18	i=20
30	1016	25	time	28,740	1440.26	597.88	1023.90	151.96	43.83	186.77	58.38	85.53	49.38	**33.03**
	1298	51	nodes		11,694,534	5,580,555	10,458,174	1,179,236	146,896	692,870	253,465	350,497	179,790	37,705
					i=8	i=10	i=12			i=8	i=10	i=12		
38	582	17	time	**62.18**	1554.65	2046.00	272.69			134.41	216.94	103.17		
	727	59	nodes		8,986,648	11,868,672	1,412,976			348,723	583,401	242,429		
					i=6	i=8	i=10	i=12		i=6	i=8	i=10	i=12	
50	479	18	time	716.60	4140.29	2493.75	66.66	52.11		78.53	36.03	**12.75**	38.52	
	517	58	nodes		28,201,843	15,729,294	403,234	110,302		204,886	104,289	25,507	5,766	

Table 3: CPU time in seconds and nodes visited to prove optimality for genetic linkage analysis.

solutions whose cost is above the optimal one, like best-first search. Notice that toolbar3 and AOEDAC+DVO are able to solve relatively efficiently only the first 3 test instances.

ISCAS'89 Circuits ISCAS'89 circuits[1] are a common benchmark used in formal verification and diagnosis. For our purpose, we converted each of these circuits into a non-binary WCSP instance by removing flip-flops and buffers in a standard way, creating hard constraints for gates and uniform unary cost functions for inputs. The penalty costs were distributed uniformly randomly between 1 and 10.

Table 2 reports the results for experiments with 6 circuits. We observe again that AOBFMB(i) is the best performing algorithm. For instance, on the s1196 circuit, AOBFMB(10) is 174 times faster and explores a search space 260 times smaller than AOBBMB(10). In summary, the best-first AND/OR search algorithms with mini-bucket heuristics cause significant time savings especially for relatively small i-bounds which generate relatively weak heuristic estimates. The performance of the toolbar3 and AOEDAC+DVO algorithms that are designed specifically for WCSP was very poor on this dataset and they were not able to solve the problems within the 1 hour time limit.

Belief Networks

The maximum likelihood haplotype problem in genetic linkage analysis is the task of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data. It is equivalent to finding the most probable explanation of a belief network that represents the pedigree data (Fishelson & Geiger 2002).

[1] Available at http://www.fm.vslib.cz/ kes/asic/iscas/

Table 3 displays the results obtained for 5 hard linkage analysis networks[2]. In addition to the depth-first and best-first AND/OR search algorithms with pre-compiled mini-bucket heuristics (i.e. AOBBMB(i), AOBFMB(i)), we also ran Superlink 1.5, which is one of the most efficient solvers for genetic linkage analysis. We notice again the superiority of AOBFMB(i) over AOBBMB(i), especially for relatively small i-bounds. On some instances (e.g. ped1, ped30), the best-first search algorithm AOBFMB(i) is several orders of magnitude faster than Superlink.

0/1 Integer Linear Programs

For this domain we experimented with random combinatorial auctions which were drawn from the regions-upv distribution of the CATS 2.0 test suite[3] and simulate the auction of radio spectrum for different geographical areas.

In combinatorial auctions, an auctioneer has a set of goods to sell and the buyers submit a set of bids over subsets of goods. The winner determination problem is to label the bids as winning or loosing so as to maximize the sum of the accepted bid prices under the constraint that each good is allocated to at most one bid. The problem can be formulated as 0/1 ILP, as described in (Leyton-Brown, Pearson, & Shoham 2000).

We consider two classes of best-first AND/OR search algorithms, as follows: AOBF which explores the context-minimal AND/OR search graph, and AOBF+PVO which explores a dynamic AND/OR search tree using partial variable orderings, respectively. We compare the best-first search

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

[3] http://cats.stanford.edu/

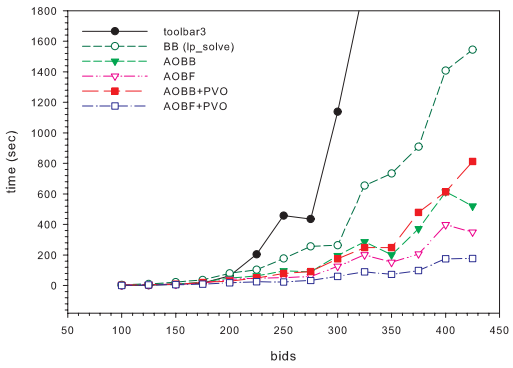

Figure 2: Results for combinatorial auctions.

algorithms against two depth-first AND/OR Branch-and-Bound algorithms for 0/1 ILPs which were recently proposed by (Marinescu & Dechter 2006a): AND/OR Branch-and-Bound with full context-based caching (AOBB), and AND/OR Branch-and-Bound with partial variable ordering (AOBB+PVO), respectively. The guiding heuristic function is computed by solving the linear relaxation of the current subproblem with the SIMPLEX method (we used the implementation from the `lp_solve5.5` library[4]).

For reference, we include results obtained with the classic OR Branch-and-Bound algorithm (BB) available from the `lp_solve` library. The algorithms BB, AOBB+PVO and AOBF+PVO used a dynamic variable selection heuristic based on *reduced costs* (or dual values) which selects the next fractional variable with the smallest reduced cost. Since combinatorial auctions can be formulated as binary WCSP instances (Dechter 2003), we also ran `toolbar3`.

Figure 2 displays the results for experiments with combinatorial auctions with 100 goods and increasing number of bids. Each data point represents an average over 10 random samples. We observe that the best-first AND/OR search algorithms (AOBF, AOBF+PVO) outperform their AND/OR Branch-and-Bound counterparts (AOBB, AOBF+PVO), especially when the number of bids increases. When looking at the two best-first algorithms we notice the superiority of AOBF+PVO over AOBF. This demonstrates the power of the dynamic variable selection heuristic which is able in this case to cut the search tree dramatically. In summary, AOBF+PVO is the best performing algorithm and, on some of the hardest instances, it outperforms its competitors with up to one order of magnitude.

Conclusion

In this paper we introduced a best-first AND/OR search algorithm which extends the classic AO* algorithm and traverses a context-minimal AND/OR search graph for solving optimization tasks in graphical models. We also proposed a best-first search algorithm that explores an AND/OR search tree, rather than a graph, and incorporates dynamic variable ordering heuristics. The efficiency of the best-first AND/OR search approach compared to the depth-first AND/OR Branch-and-Bound search is demonstrated empirically on various benchmarks including random as well as real-world problem instances.

Our approach leaves room for further improvements. The space required by AOBF can be enormous, due to the fact that all the nodes generated by the algorithm have to be saved prior to termination. Therefore, AOBF can be extended to incorporate a memory bounding scheme similar to the one suggested in (Chakrabati *et al.* 1989).

[4]http://groups.yahoo.com/group/lp_solve/

Acknowledgments

This work was supported by the NSF grant IIS-0412854.

References

Bensana, E.; Lemaitre, M.; and Verfaillie, G. 1999. Earth observation satellite management. *Constraints* 4(3):293–299.

Chakrabati, P.; Ghose, S.; Acharya, A.; and de Sarkar, S. 1989. Heuristic search in restricted memory. *In Artificial Intelligence* 3(41):197–221.

de Givry, S.; Heras, F.; Larrosa, J.; and Zytnicki, M. 2005. Existential arc consistency: getting closer to full arc consistency in weighted csps. *In IJCAI*.

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. 2003. *Constraint Processing*. MIT Press.

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

Freuder, E., and Quinn, M. 1985. Taking advantage of stable sets of variables in constraint satisfaction problems. *In IJCAI* 1076–1078.

Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a universal test suite for combinatorial auction algorithms. *In ACM Electronic Commerce* 66–76.

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

Marinescu, R., and Dechter, R. 2006a. And/or branch-and-bound search for pure 0/1 integer linear programming problems. *In CPAIOR* 152–166.

Marinescu, R., and Dechter, R. 2006b. Dynamic orderings for and/or branch-and-bound search in graphical models. *In ECAI* 138–142.

Marinescu, R., and Dechter, R. 2006c. Memory intensive branch-and-bound search for graphical models. *In AAAI*.

Nemhauser, G., and Wolsey, L. 1988. *Integer and combinatorial optimization.* Wiley.

Nillson, K. 1980. *Principles of Artificial Intelligence.* Tioga.

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