AND/OR Branch-and-Bound for Solving Mixed Integer Linear Programming Problems

Student:Radu Marinescu
Supervisor: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 paradigm 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. In this paper we extend the recently introduced AND/OR Branch-and-Bound algorithm [1] for solving 0/1 Mixed Integer Linear Programming problems. We propose a *static* version based on pseudo-trees, as well as a *dynamic* one based on hypergraph separators. Preliminary evaluation on problem instances from MIPLIB2003 shows promise that the new schemes are likely to improve over the traditional methods.

1 Introduction

Graphical models (e.g. constraint networks, belief networks) are a powerful representation framework for automated reasoning tasks. These models use graphs to capture conditional independencies between variables, allowing for a concise representation of the knowledge. Optimization tasks defined within this framework are typically tackled with either *search* (e.g. branch-and-bound) or *inference* (e.g. variable elimination). Search methods are time exponential in the number of variables and can operate in linear space. Inference algorithms are time and space exponential in the *tree width* of the problem. This potentially high space complexity makes the latter methods impractical in many cases.

The AND/OR search space for graphical models [2] is a newly introduced framework for search that is sensitive to the independencies in the model, often resulting in exponentially reduced complexities. The AND/OR search is based on a pseudo-tree which expresses independencies between variables, resulting in a search tree exponential in the depth of the pseudo-tree, rather than the number of variables.

The 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 [1]. In this paper we extend the algorithm for solving combinatorial optimization problems from the class of 0/1 Mixed Integer Linear Programs (MILP) [3]. First, we present the *static* version of the algorithm guided by a pseudo-tree arrangement of the constraint graph. Second, we propose a *dynamic* version of AOBB which uses a dynamic decomposition of the problem, based on hypergraph separators. Our preliminary evaluation of several hard problem instances from the MIPLIB2003 library shows promise that the new methods can improve significantly over the traditional OR tree search algorithms.

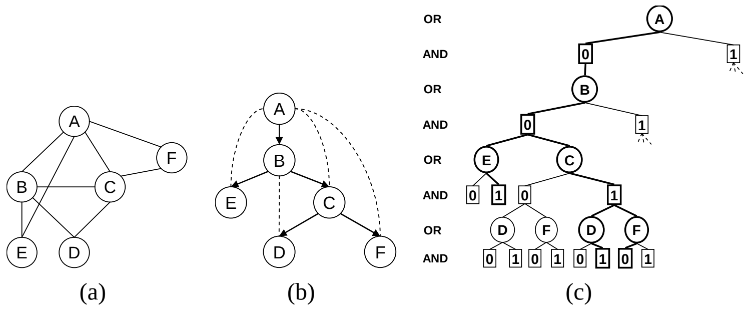

Fig. 1. An AND/OR search tree.

2 Background

2.1 Constraint Optimization Problems

A finite *Constraint Optimization Problem* (COP) is a six-tuple $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F}, \otimes, \Downarrow, Z)$, 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 constraints. Constraints can be either *soft* (cost functions) or *hard* (sets of allowed tuples). 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. $\otimes_i f_i$ is a *combination* operator, $\otimes_i f_i \in \{\prod_i f_i, \sum_i f_i\}$ and $\Downarrow_Y f$ is an *elimination* operator, $\Downarrow_Y f \in \{max_{S-Y} f, min_{S-Y} f\}$, where S is the scope of function f and $Y \subseteq \mathcal{X}$. The scope of $\Downarrow_Y f$ is Y. The *constraint graph* of a COP has a node for each variable and an arc connects any two variables appearing in the same function's scope. An optimization task is defined by $g(Z) = \Downarrow_Z \otimes_{i=1}^{m} f_i$, where $Z \subseteq \mathcal{X}$. A *global optimization* is the task of finding the best global cost, namely $Z = \emptyset$.

2.2 Mixed Integer Linear Programming

A *Mixed Integer Linear Programming* (MILP) problem is a linear program where some of the decision variables are constrained to have only integer values at the optimal solution. Hence, we define the MILP problem as follows:

$$min \text{ or } max\{c^T x \mid Ax \leq b, x \geq 0, x \text{ has integer components}\}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$. Here c represents the cost vector and x is the vector of variables. The vector b and the matrix A define the m linear constraints. An important special case is a decision variable x_i that is integer with $0 \leq x_i \leq 1$. This forces x_i to be either 0 or 1 at the solution. Variables like x_i are called *0/1* or *binary integer variables*. Subsequently, a MILP problem with binary integer variables is also called a *0/1 Mixed Integer Linear Programming* problem (for more details see [4, 3]).

2.3 AND/OR Search Spaces

In this section we briefly review the notion of AND/OR search spaces for COPs [2, 1]. Given a COP instance, the AND/OR search space is defined using a backbone *pseudo-*

tree arrangement of the constraint graph. A pseudo-tree arrangement T of a graph G [5, 6] is a rooted tree with the same set of vertices as G and the property that adjacent vertices from G must be in the same branch of T.

Given a COP instance, its constraint 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 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, $path(x_i) = (\langle X_1, x_1 \rangle, ..., \langle X_{i-1}, x_{i-1} \rangle)$. 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. The size of the resulting search tree is bounded exponentially by the depth of the pseudo-tree, which in practice may be far smaller than the number of variables.

Example 1. Figure 1(a) shows a binary COP instance. Figure 1(b) shows a pseudo-tree of the constraint graph, together with the back-arcs (dotted lines). Figure 1(c) shows a partial AND/OR search tree based on the pseudo-tree, for bi-valued variables (for AND nodes we only denote the value, namely $\langle A, 0 \rangle$ is written as $\boxed{0}$ child of A).

3 Static AND/OR Branch-and-Bound

In [1] we introduced a novel Branch-and-Bound algorithm, called AOBB, that explores the AND/OR search space for solving optimization tasks in graphical models. The idea was supported by an extended empirical evaluation which was concentrated on two common optimization problems, solving Weighted CSPs [7] and finding the Bayesian MPE in belief networks [8], and demonstrated clearly the impact of the AND/OR tree search over the traditional OR tree search.

3.1 AOBB for Constraint Optimization Problems

In this section we assume without loss of generality a COP instance $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$ with *summation* and *minimization* as combination and elimination operators, and a global optimization function defined by $f(\mathcal{X}) = min_{\mathcal{X}} \sum_{i=1} f_i$.

Given a pseudo-tree arrangement of the constraint graph of $\mathcal{P}$, algorithm AOBB traverses the corresponding AND/OR search tree in a depth-first manner, expanding alternating levels of OR and AND nodes. The arcs from X_i to $\langle X_i, x_i \rangle$ are associated with appropriate *labels* of the constraints in $\mathcal{P}$. The algorithm maintains *values* for each node, accumulating the result of the computation resulted from the subtree below. The value $v(n)$ of a node n represents the optimal cost solution to the subproblem rooted at n, subject to the current partial instantiation along the path from the root. In particular, the value of the root node represents the cost of the optimal solution to the initial problem. OR nodes compute their value by minimizing the values of their children, while AND nodes by summation.

In addition to their values, each node at the search frontier is also assigned a *static* heuristic lower-bound estimate $h(n)$ of $v(n)$, which is used for choosing the most promising extension of the current partial solution subtree being explored and pruning irrelevant portions of the search space. In [1] we discussed several approaches for computing h, based on approximate inference as well as soft directional arc-consistency.

During search, each node n along the *active path* (i.e. current path from the root of the search tree) roots a partially explored solution subtree of the corresponding subproblem, called *active partial subtree*. Given an active partial tree $\mathcal{APT}(n)$, we showed in [1] that it is possible to compute recursively a tighter *lower bound* $lb(n)$ on $v(n)$, based on the static heuristic functions $h(m)$ of the nodes $m \in \mathcal{APT}(n)$ at the search frontier and the portion of the search space below n that has already been explored. In addition, the current best cost solution subtree rooted at n provides an *upper bound* $ub(n)$ on $v(n)$. Consequently, we showed that searching below the tip node of the active path can be safely terminated as soon as the updated lower bound exceeds the upper bound for some node along the active path.

3.2 AOBB for 0/1 Mixed Integer Linear Programming

In general, we can view any linear program as a COP instance with linear constraints (inequalities and equalities) and a linear objective function. Therefore, a 0/1 MILP problem can be defined as a quadruple $\mathcal{P}_{MILP} = (\mathcal{X}, \mathcal{X}_I, \mathcal{C}, f)$, where $\mathcal{X} = \{X_1, ..., X_n\}$ is the set of variables, $\mathcal{X}_I \subseteq \mathcal{X}$ is the subset of binary integer variables, $\mathcal{C} = \{C_1, ..., C_m\}$ is the set of constraints and $f(\mathcal{X}) = \sum_{i=1}^{n} c_i X_i$ is the objective function which has to be optimized (i.e minimized or maximized). In the following we consider a minimization problem defined by $f(\mathcal{X}) = min_{\mathcal{X}} \sum_{i=1}^{n} c_i X_i$.

Given a 0/1 MILP problem $\mathcal{P}_{MILP} = (\mathcal{X}, \mathcal{X}_I, \mathcal{C}, f)$ with constraint graph G, the corresponding AND/OR search tree is based on a *start pseudo-tree* T' of G. The start pseudo-tree has the following properties: (i) it has the same root and is a subgraph of some pseudo-tree T of G; (ii) the nodes of T' are all in $\mathcal{X}_I$, namely they correspond to the integer variables of $\mathcal{P}_{MILP}$.

The algorithm presented in Section 3.1 can be easily modified to solve 0/1 MILPs. In this case, the value $v(n)$ of a node n in the search tree is the minimal cost solution to the subproblem $\mathcal{P}_n$ rooted at n. The node n can be an OR node labeled with $n = X_i$ or an AND node labeled with $n = \langle X_i, x_i \rangle$. In either case, $\mathcal{P}_n$ is defined by the set of constraints and constraint projections involving only the descendants $desc_T(X_i)$ of X_i in T, subject to the current instantiation along the active path, and the local cost function that corresponds to the projection on $desc_T(X_i)$ of the global $f(\mathcal{X})$. The static lower-bounding heuristic estimate $h(n)$ is computed by solving the linear relaxation (i.e. relaxing the integrality restrictions) of the respective subproblem.

For illustration, consider a 0/1 MILP problem having the constraint graph in Figure 1(a), where every edge represents a linear inequality or equality between the corresponding pair of variables. Variables $\{A, B, C, E\}$ are integers restricted to the values 0 and 1. The objective is to minimize the cost function $f(\mathcal{X}) = 6A + 4B + 5C + 3D + 7E + 9F$. The pseudo-tree is given in Figure 1(b). The subproblem rooted at node C in the search tree corresponds to minimizing the cost function $f_C = 5C + 3D + 7E$, subject to the constraints and constraint projections involving only the variables $\{C, D, E\}$. Notice that the search algorithm needs only to explore the start pseudo-tree represented by the integer variables.

problem	n	m	n_I	h
dcmulti	548	290	75	**44**
egout	141	98	55	54
enigma	100	21	100	98
lseu	89	28	89	**72**
p0033	33	16	33	**22**
p0040	40	23	40	**28**
p0201	201	133	201	**160**
p0282	282	241	282	**196**
pk1	86	45	55	54
pp08a	240	136	64	63

Table 1. MIPLIB2003 problem instances.

In Table 1 we assess the structural properties of 10 real-world 0/1 MILP problem instances from the MIPLIB2003[1] benchmarks library. For each test case we provide the total number of variables (n), the number of constraints (m), the number of integer variables (n_I) and the depth of the start pseudo-tree generated (h). We observe that in many cases $h \ll n_I$ (e.g. dcmulti, lseu, p0033, etc.). Since AOBB has a worst-case time complexity bounded exponentially by the depth of the pseudo-tree rather than the number of variables, we would expect impressive time savings on those problem instances, as compared to the traditional OR tree search approach.

4 Dynamic AND/OR Branch-and-Bound

The AND/OR Branch-and-Bound algorithm discussed so far was guided by a *static* variable ordering induced by a pseudo-tree arrangement of the underlying constraint graph of the problem. We now propose a new AND/OR Branch-and-Bound algorithm that uses a *dynamic* variable ordering generated by a dynamic decomposition of the problem, based on hypergraph separators.

4.1 Hypergraph Separator Decomposition

Given a COP instance $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, its hypergraph representation $\mathcal{H} = (V, E)$ is a hypergraph whose vertex set V consists of the constraints in $\mathcal{F}$, and there is a hyperedge for each variable in $\mathcal{X}$ connecting all the constraints that contain that variable.

A *hypergraph separator decomposition* is a triple $(\mathcal{H}, \mathcal{S}, \mathcal{R})$ where: (i) $\mathcal{S} \subset E$, and the removal of $\mathcal{S}$ separates $\mathcal{H}$ into k disconnected components (subgraphs) $\mathcal{H}_1, ..., \mathcal{H}_k$; (ii) $\mathcal{R}$ is a relation over the size of the disjoint subgraphs. Because the problem of computing an optimal partition of a hypergraph is NP-complete, a multi-level hypergraph partition algorithm package, such as `hMETIS` [9], can be used to find separators.

4.2 Dynamic Decomposition and AOBB

In this section we discuss briefly how to integrate hypergraph separator decomposition into the AOBB algorithm for solving 0/1 MILP problems. Finding hypergraph separa-

[1] Available at http://miplib.zib.de/

tors naturally leads to a divide-and-conquer strategy. The separator becomes the root of the corresponding tree structure, while the subtrees become the subproblems induced by the separator.

The dynamic version of the AOBB algorithm works as follows. It takes as input the problem instance $\mathcal{P}_{MILP}$, the corresponding constraint graph G and the separator $\mathcal{S}$ of G, whose initial value is $\emptyset$. The algorithm expands alternating levels of OR and AND nodes in a similar manner as its static predecessor. When expanding a node n, the algorithm computes the static heuristic estimate $h(n)$ of $v(n)$ by solving the linear relaxation of the subproblem rooted at n. This process may result in one or more integer variables have their values determined. These variables can therefore be removed from the subproblem. Consequently, a new separator $\mathcal{S}$ for the respective subproblem can be computed, based on a simplified constraint graph. Then, the first variable from $\mathcal{S}$ is chosen for instantiation and the search continues. When computing a separator, care must be taken to ensure that only integer variables belong to the separator.

5 Conclusion

In this paper we extended the AND/OR Branch-and-Bound algorithm for solving the class of 0/1 Mixed Integer Linear Programming problems. The contribution of the paper is two-fold. First, we restricted the algorithm to a static variable ordering induced by a start pseudo-tree of the constraint graph. Second, we allowed the algorithm to use a dynamic variable ordering based on hypergraph separators. Preliminary assessment of the structural properties of several hard problem instances from the MIPLIB2003 library showed promise that the new AND/OR search schemes can improve significantly over the traditional OR tree search approach. Finally, we mention that more advanced strategies developed in the recent years for integer programming, such as *branch-and-cut* [3], can be readily adapted to exploit the AND/OR structural paradigm.

References

1. R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. *Proceedings of the International Joint Conference on Artificial Inteligence*, 2005.
2. R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks and their and/or search spaces. *Proceedings of Uncertainty in Artificial Inteligence*, 2004.
3. G.L. Nemhauser and L.A. Wosley. *Integer and Combinatorial Optimization*. New York, Wiley, 1988.
4. E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.
5. E. Freuder and M. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. *Proc. of the International Joint Conference on Artificial Inteligence*, 1985.
6. R. Bayardo and D. Miranker. On the space-time trade-off in solving constraint satisfaction problems. *Proc. of the International Joint Conference on Artificial Inteligence*, 1995.
7. S. Bistarelli, U. Montanari, and F. Rossi. Semiring based constraint solving and optimization. *Journal of ACM*, 44(2):309–315, 1997.
8. J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan-Kaufmann, 1988.
9. hMETIS. http://www-users.cs.umn.edu/ karypis/metis/hmetis.