AFRL-OSR-VA-TR-2014-0281

A Unified Approach to Optimization

John Hooker
CARNEGIE MELLON UNIVERSITY

10/02/2014
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
27-09-2014	Final report	15-07-2011 to 14-07-2014

4. TITLE AND SUBTITLE

Unified Approach to Optimization

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA95590-11-1-0180

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
John Hooker (PI)
Tallys Yunes

5d. PROJECT NUMBER
25998

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
25998.1.1150100-FIN

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
876 North Randolph St, Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution A - Approved for public release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The purpose of this research is to identify underlying unity in optimization methods, and to use the resulting insights to combine methods so as to exploit complementary strengths. In particular, we propose strategies for the integration of mathematical programming, constraint programming, dynamic programming, and heuristic methods. We report achievements in seven specific approaches to unification: high-level modeling, finite-domain cuts, bounds from decision diagrams, primal heuristics from decision diagram, decision diagrams and dynamic programming, logic-based Benders decomposition, and unification of exact and heuristic methods.

15. SUBJECT TERMS

Optimization, constraint programming

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		15	Ted Curran
					19b. TELEPHONE NUMBER *(include area code)* (412) 268-8118

Unified Approach to Optimization
Final Report

AFOSR Grant FA9550-11-1-0180
Program manager: Dr. Fariba Fahroo
Grant period: 15 July 2011 to 14 July 2014

John Hooker[1] (PI) and Tallys Yunes[2]

September 2014

The purpose of this research is to identify underlying unity in optimization methods, and to use the resulting insights to combine methods so as to exploit complementary strengths. In particular, we propose strategies for the integration of mathematical programming, constraint programming, dynamic programming, and heuristic methods. We report achievements in seven specific approaches to unification: high-level modeling, finite-domain cuts, bounds from decision diagrams, primal heuristics from decision diagrams, decision diagrams and dynamic programming, logic-based Benders decomposition, and unification of exact and heuristic methods.

The publications associated with this research appear in the reference list at the end of the report. We note that some of our results appear in the doctoral dissertations [11] and [12], the former of which received the 2014 Doctoral Thesis Award of the Association for Constraint Programming. A chapter of the latter received the 2014 Student Paper Award of the INFORMS Computing Society.

1 Modeling

The aim of our modeling research is to design a modeling framework that is conducive to integrating technologies. A key to integrated modeling is to formulate a problem with high-level *metaconstraints*, which are inspired by the "global constraints" of constraint programming (CP). A metaconstraint enforces a structured set of more elementary contraints. For example, scheduling constraints that require that jobs run one at a time can be enforced with a single noOverlap constraint. Metaconstraints have the advantage that they convey problem substructure to the solver. This contrasts with the atomistic modeling style of mixed integer programming (MIP) and satisfiability (SAT) solvers, which relies on the solver to detect structure in an undifferentiated collection of inequality or clausal constraints.

Metaconstraints enable integrated modeling because a single high-level constraint can invoke search, inference, and relaxation techniques from several technologies. In fact, an important commonality of optimization methods is that they all rely on search, inference, and relaxation, with CP emphasizing inference and MIP emphasizing relaxation. Thus a noOverlap constraint can invoke inference methods from CP technology, such as edge-finding rules and other filtering techniques. It can generate polyhedral relaxations and cutting planes from MIP. It can also help guide the search by suggesting a branching rule that can be applied when the constraint is violated. We systematically developed the advantages of metaconstraint-based modeling in a long book chapter [1], which

[1]Carnegie Mellon University
[2]University of Miami

contrasts CP and MIP modeling on the one hand with integrated modeling on the other for a wide variety of problems.

Our work on metaconstraints raised a fundamental issue that must be resolved before integrated modeling can be successful. Metaconstraints invoke relaxations and reformulations that typically introduce auxiliary variables. The problem is that variables created by different metaconstraints may in fact be identical, or otherwise related, and this must be recognized if one is to have a tight relaxation and effective filtering. We address this problem with *semantic typing* of variables. The modeler assigns semantic types to the original variables, and the modeling system automatically assigns semantic types to auxiliary variables. The types given to variables indicate whether they are equivalent or otherwise related. An additional advantage of semantic typing is that it imposes a natural structure and organization on the model, allowing it to be more readable and providing consistency checks.

Semantic typing is implemented by organizing a model around *predicates*, which denote relations among variables, much as in a relational database. A variable is declared by associating it with a predicate and a keyword that acts analogously to a database query. To take a very simple example, consider an assignment problem in which x_i is the job assigned to worker i, y_j is the worker assigned to job j, and c_{ij} is the cost of assigning i to j. There are restrictions on which workers are assigned to a given job, and vice-versa, as well as other constraints. The problem can be written

$$\begin{aligned}
&\min \ \sum_i c_{ix_i} \\
&\text{alldiff}(x_1, \ldots, x_n), \ \ \text{alldiff}(y_1, \ldots, y_n) \\
&x_i \in X_i, \text{ all } i, \ \ y_j \in Y_j, \text{ all } j \\
&\text{additional constraints}
\end{aligned}$$

where the alldiff constraints require the variables listed to take distinct values. The modeling system might generate a classical assignment problem relaxation for the first alldiff:

$$\sum_j \delta_{ij} = 1, \text{ all } i, \ \ \sum_j \delta_{ij} = 1, \text{ all } j$$

where 0-1 variable $\delta_{ij} = 1$ when $x_i = j$, and similarly for the second alldiff:

$$\sum_j \delta'_{ij} = 1, \text{ all } i, \ \ \sum_j \delta'_{ij} = 1, \text{ all } j$$

The objective might also be written in terms of 0-1 variables:

$$\min \ \sum_{ij} c_{ij} \delta''_{ij}$$

The auxiliary 0-1 variables δ_{ij}, δ'_{ij}, and δ''_{ij} are actually equivalent and should be replaced by the same variable in the relaxation. This is accomplished by semantic typing as follows.

To begin with, the model is organized around a predicate that matches workers and jobs, such as `assign(worker,job)`. The variables are declared

```
x[i] is which job assign(worker i)
y[j] is which worker assign(job j)
```

where reserved words are underlined. Because x_i and y_j are related to the same predicate by a `which` keyword, the modeling system generates *channeling constraints* to relate the two variables:

$$x_{y_j} = j, \text{ all } j; \ \ y_{x_i} = i, \text{ all } i$$

The system also generates a type declaration for the auxiliary variables as they are introduced. Because they are related to the same predicate in the same way, the variables δ_{ij}, δ'_{ij} and δ''_{ij} are given the same declaration:

```
whether assign(worker i, job j)
```

Since these variables receive the same type, they are identified in the relaxation, as desired.

Semantic typing is developed in our paper [2], which illustrates the idea for piecewise linear modeling, employee scheduling, ad placement, latin squares, disjunctions of linear systems, temporal modeling with interval variables, and traveling salesman problems with side constraints.

In future work, we intend to code a modeling system that will be freely distributed and allow practitioners to convert a model written with semantic typing to problem specifications that can be input to popular solvers.

2 Finite-Domain Cuts

This arm of our research combines the finite-domain modeling approach of CP with polyhedral analysis of MIP. It begins with the fact that CP typically formulates combinatorial problems with finite-domain variables, while MIP uses 0-1 variables. This suggests the possibility of conducting polyhedral analysis of the convex hull of the feasible set in the finite-domain space, rather than in the 0-1 space as in MIP. The finite-domain cutting planes that result can then be mapped into 0-1 space to be combined with traditional 0-1 cuts.

We found that for at least one classical problem, this procedure results in tighter bounds than can be obtained from all known 0-1 cuts, and the bounds are calculated in much less time because fewer cuts are necessary. Our initial results appear in [3], while [4] provides a fuller mathematical treatment that proves theoretically the superiority of finite-domain cuts. We also provided polynomial-time separation algorithms for the cuts.

We focused on the classical graph coloring problem, which asks how a minimum number of colors can be assigned to vertices of a graph so that adjacent vertices receive different colors. CP formulates the problem using multiple all-different constraints and finite-domain variables x_i that indicate the color that is assigned to vertex i of a graph. For any clique V_k of the graph, the variables x_i for $i \in V_k$ are required to take different values. MIP formulates the problem using 0-1 variables y_{ij} that indicate whether vertex i receives color j. The MIP model is

$$\begin{aligned} &\min \ \sum_j w_j \\ &\sum_j y_{ij} = 1, \ \ \text{all } i \\ &\sum_{i \in V_k} y_{ij} \le w_j, \ \ \text{all } j,k \end{aligned}$$

where 0-1 variable $w_j = 1$ when color j is used, and k indexes all cliques V_k of some clique cover. A cut in the x_i-space is mapped to a cut in 0-1 space simply by replacing each x_i by $\sum_j j y_{ij}$.

We studied cuts based on cycles, webs, and paths in the graph. The path cuts are redundant of known 0-1 cuts, but cycles and webs yield facet-defining finite-domain cuts that produce tighter bounds that known 0-1 cuts. Proving these results required that we advance beyond the polyhedral proof techniques that are used for conventional 0-1 cuts. Figure 1 illustrates a cycle of five all-different constraints, each indicated by a solid oval. For example, V_1 corresponds to the constraint

$$\text{alldiff}(x_0, x_1, x_2, x_3, x_{10}, x_{11})$$

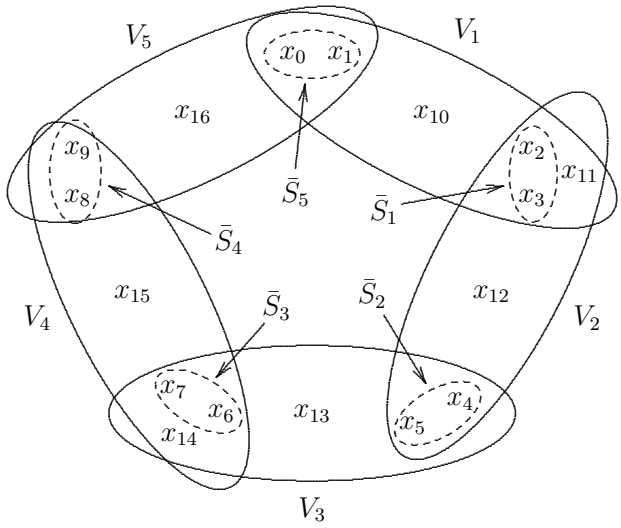

Figure 1: A cycle of all-different constraints that gives rise to finite-domain cuts. The solid ovals correspond to constraints alldiff(V_k) for $k = 1, \ldots, 5$.

This cycle gives rise to the valid cuts

$$x_1 + \cdots + x_9 \geq 20$$
$$z \geq \tfrac{1}{10}(x_1 + \cdots + x_9) + 2$$

where z indicates the largest color number. These two cuts alone provide a tighter bound than the corresponding classical clique inequality combined with all 320 odd-hole cuts for this cycle. We obtained similar results for webs (Fig. 2).

We tested the method on 23 DIMACS benchmark instances. For finite-domain cuts we used only a subset of the cycle cuts, identified heuristically. We compared them with a collection of all classical odd-hole cuts using the clique cover formulation above, which is the tightest known MIP formulation. Even though graph coloring is one of the most intensely studied combinatorial problems, we obtained tighter bounds in 8 of the instances. More importantly, we obtained nearly all of the bounds more rapidly, in some cases one or two orders of magnitude more rapidly.

These results suggest that finite-domain cuts could profitably augment the linear relaxation currently used in MIP solvers. They could combine with or replace the traditional cuts to obtain better bounds in less time. Because we provide fast and complete separation algorithms, the solution of the current linear relaxation can be used to generate separating cuts in very little time at deeper nodes in the search tree.

3 Bounds from Decision Diagrams

In this research we investigate a second technique for obtaining optimization bounds from CP, namely from relaxed decision diagrams. Decision diagrams have been used historically for circuit design and verification, as well as product configuration. We built on previous work that adapted decision diagrams to constraint solving and optimization, in particular as a device for strengthening

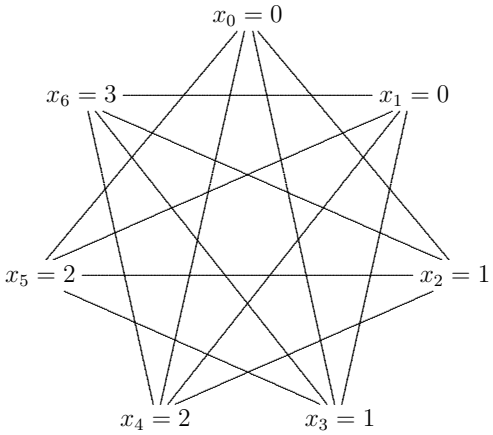

Figure 2: Web $W(7, 2)$, which is an odd antihole. Variables connected by an edge appear in a common alldiff constraint. A feasible solution is shown.

propagation and filtering in CP solvers. The key to this work is to represent the feasible set with a relaxed decision diagram of limited width.

In an optimization problem, a relaxed decision diagram can provide a bound on the optimal value equal to the length of a longest (or shortest) path through the diagram. One can obtain a bound of any desired tightness by increasing the maximum width of the relaxed diagram, recognizing that a wider decision diagram requires more computation time. We found that for at least one classical problem, decision diagrams can provide tighter bounds, in much less time, than the full cutting plane resources of a state-of-the-art MIP solver. These results are presented in [7], which relies on results in [5, 6].

We focused on the stable set problem, which is defined on a graph in which the vertices are assigned weights. The problem is to find a maximum-weight subset of vertices for which no two vertices are adjacent. Figure 3 shows an exact and relaxed binary decision diagram (BDD) representing a small stable set problem. Here the binary variable $x_i = 1$ when vertex i is included in the stable set. A solid arc corresponds to setting $x_i = 1$, and a dashed arc to $x_i = 0$. The top-to-bottom paths in the exact BDD correspond exactly to the possible stable sets. Paths in the relaxed BDD correspond to a superset of the possible stable sets. The relaxed BDD has width 1 in this case because there is at most one node per layer. If the solid arcs are given lengths equal to the corresponding vertex weights, a longest path through the relaxed BDD provides an upper bound on the maximum weight of a stable set.

Figures 4 and 5 compare the quality of bounds obtained from decision diagrams and a commercial MIP solver for random and DIMACS benchmark instances, respectively. A lower curve indicates better bounds. Average relative bounds are plotted against the density of the graph. The graphs show that decision diagrams of almost any reasonable width provide tighter bounds for random instances than can be obtained at the root node of a state-of-the-art MIP solver, except perhaps for the sparsest graphs, even though the MIP solver benefits from cutting plane and presolve techniques that have been developed over a 50-year period. The results for DIMACS instances are similar. For these instances, the smallest BDDs (width 100) often provide weaker bounds, but width-1000 BDDs still provide tighter bounds than MIP, and much more rapidly.

In fact, the best news is that decision diagrams require less computation than MIP to obtain bounds. Only the diagrams of maximum width 10000 require computation time similar to that of

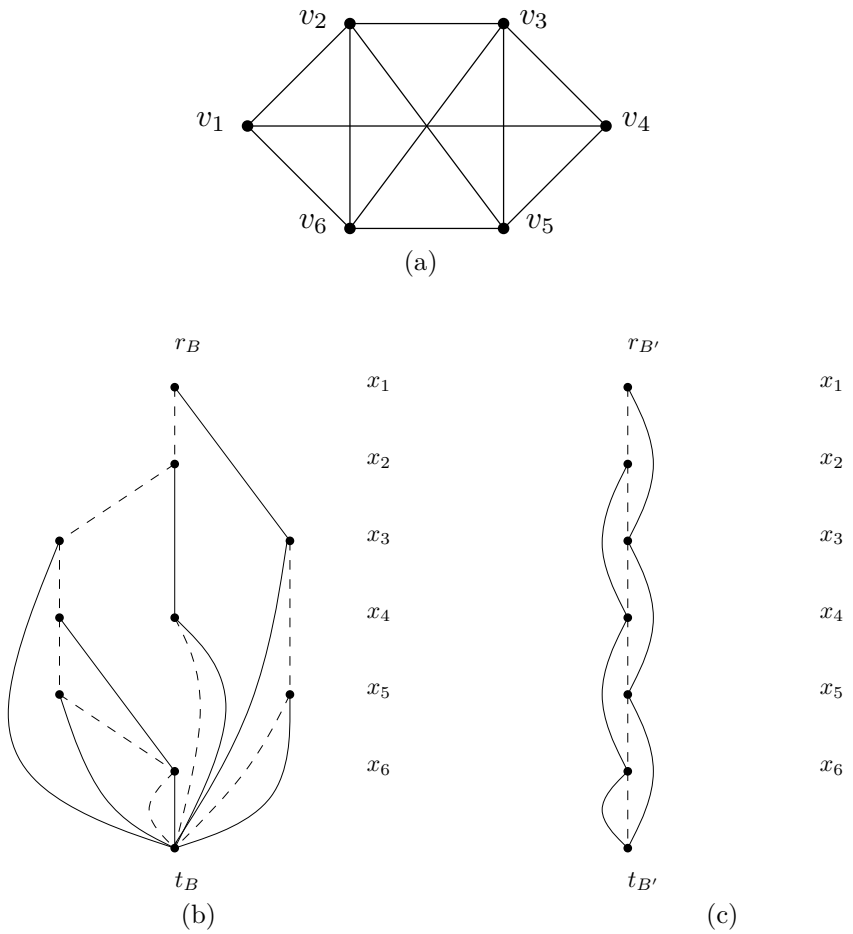

Figure 3: (a) Instance of the stable set problem. (b) Exact BDD for the instance. (c) Relaxed BDD for the instance.

the MIP solver, while diagrams of width 1000 can be processed at least an order of magnitude more rapidly. This suggests that decision diagrams can significantly strengthen the bounds used in an MIP solver while adding very little to computation time.

4 Primal Heuristics from Decision Diagrams

BDDs can also be *restricted*, as opposed to relaxed, in order to obtain valid primal bounds (feasible solutions) to optimization problems. This is achieved during the construction of the BDD by dropping nodes from layers that exceed the maximum allowed width, resulting in a BDD that includes a subset of all feasible solutions to the problem it represents. We used restricted BDDs to solve two classical binary optimization problems, set covering and set packing, but our approach is applicable to binary optimization in general. These results are presented in [8].

Our experiments were performed on a set of randomly generated instances. For both the set covering and set packing problem, we considered combinatorial instances (all costs equal to 1) as

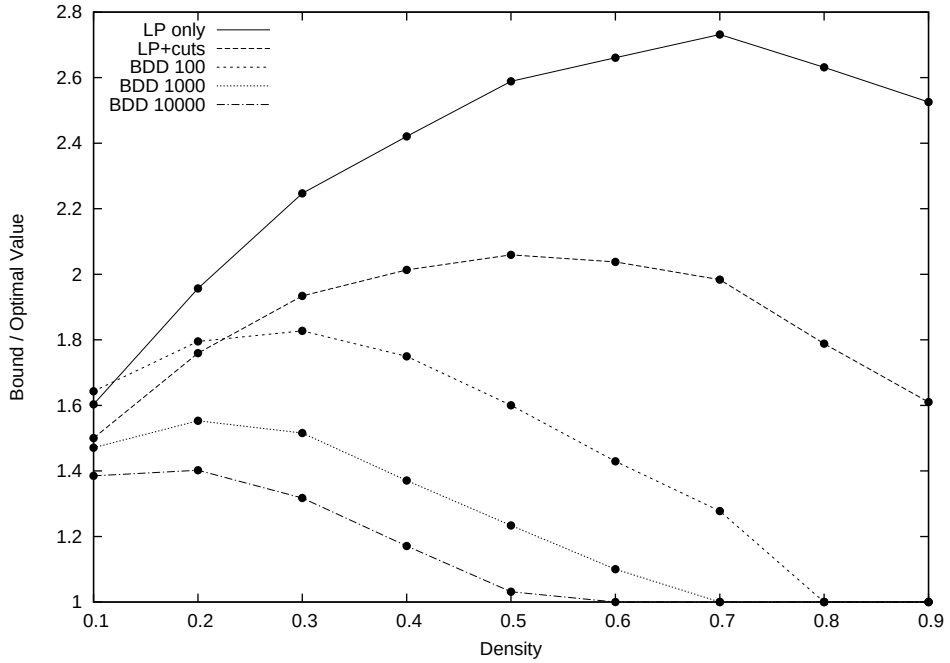

Figure 4: Bound quality vs. graph density for random instances, showing results for LP only, LP plus cutting planes, and BDDs with maximum width 100, 1000, and 10000. Each data point is the geometric mean of 20 instances.

well as weighted instances with arbitrary costs. Our restricted-BDD heuristic was compared against the heuristic capabilities of a state-of-the-art commercial MIP solver, which implements several different general-purpose heuristic methods.

For the set covering problem, solutions obtained by the restricted BDD can be up to 30% better on average than solutions obtained by the MIP solver. This advantage progressively decreases as either the bandwidth of the coefficient matrix A increases, or its sparsity decreases. In general, the BDD performs better on weighted instances. In terms of execution time, the BDD approach has a slight advantage over the MIP approach on average, and can be up to twice as fast.

For the set packing problem, the BDD approach exhibits even better performance on both the combinatorial and weighted instances. Its solutions can be up to 70% better on average than the solutions obtained by the MIP solver, with the BDD performing better on weighted instances than on combinatorial instances once again. Unlike what happened in the set covering case, BDD solutions were always at least as good as the ones produced by the MIP solver. In addition, the BDD's performance appears to improve as the bandwidth of A increases. As the sparsity of A changes, the BDD's performance is good for sparse instances, drops at first as sparsity starts to increase, and tends to slowly increase again thereafter. In terms of execution time, the BDD approach can be up to an order of magnitude faster than the traditional MIP approach.

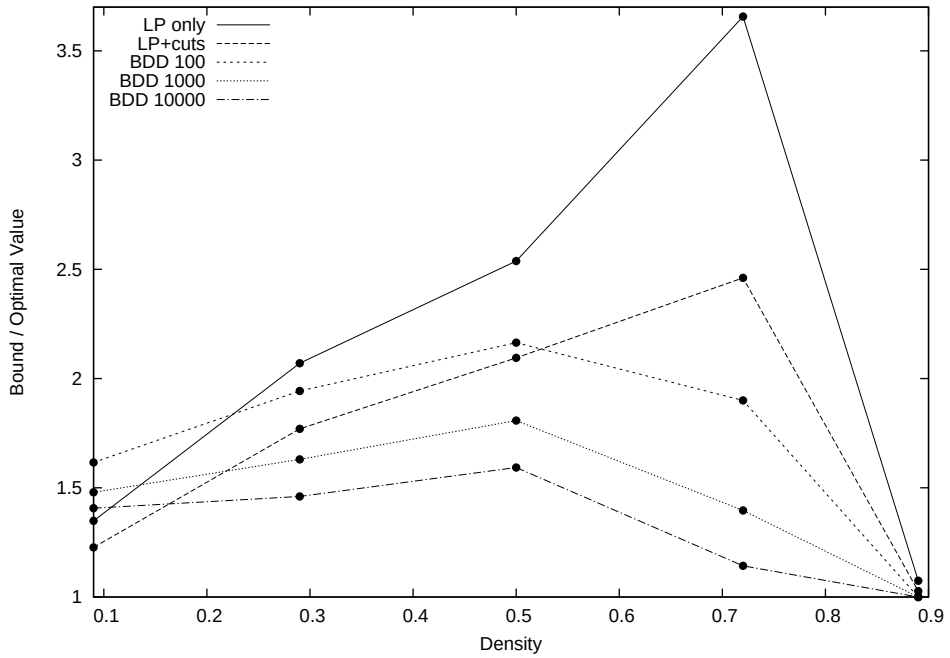

Figure 5: Bound quality vs. graph density for DIMACS instances, showing results for LP only, LP plus cutting planes, and BDDs with maximum width 100, 1000, and 10000. Each data point is the geometric mean of instances in a density interval of width 0.2.

5 Decision Diagrams and Dynamic Programming

The aim of this research is to combine decision diagrams and dynamic programming so as to obtain a more powerful solver that uses recursive models but does not solve them recursively. We believe this approach could generate a new research stream in dynamic programming, because recursive models that were previously intractable can now be attacked with BDD technology.

We arrived at this synthesis while building a general-purpose solver for discrete optimization that is based on decision diagrams, described in [9]. The solver uses branch-and-bound search, but it obtains bounds from decision diagrams, as described earlier, rather than from linear relaxations. It also uses restricted decision diagrams for primal heuristics. Most significantly, it employs a completely novel branching method that branches within a relaxed decision diagram. Briefly, it identifies the lowest layer of the diagram that is exact, meaning that the relaxed diagram is indistinguishable from an exact diagram down to that layer. It then branches on the nodes of that layer by creating a new relaxed decision diagram rooted at each node, and continues in this fashion.

This is related to dynamic programming (DP) for the following reason, as explained in [10]. A decision diagram is essentially a DP state transition graph, and a relaxed decision diagram merges some of the DP states. In fact, we build a decision diagram by associating nodes with states. This means that we model a problem by stating a DP recursion, along with a rule for merging states to create a relaxation. However, we do not solve the model by searching the state space, as is done in DP, because the recursion is used only to obtain bounds and primal heuristics. Rather, we solve the problem by branch and bound. As a result, it is of little concern whether the state space explodes

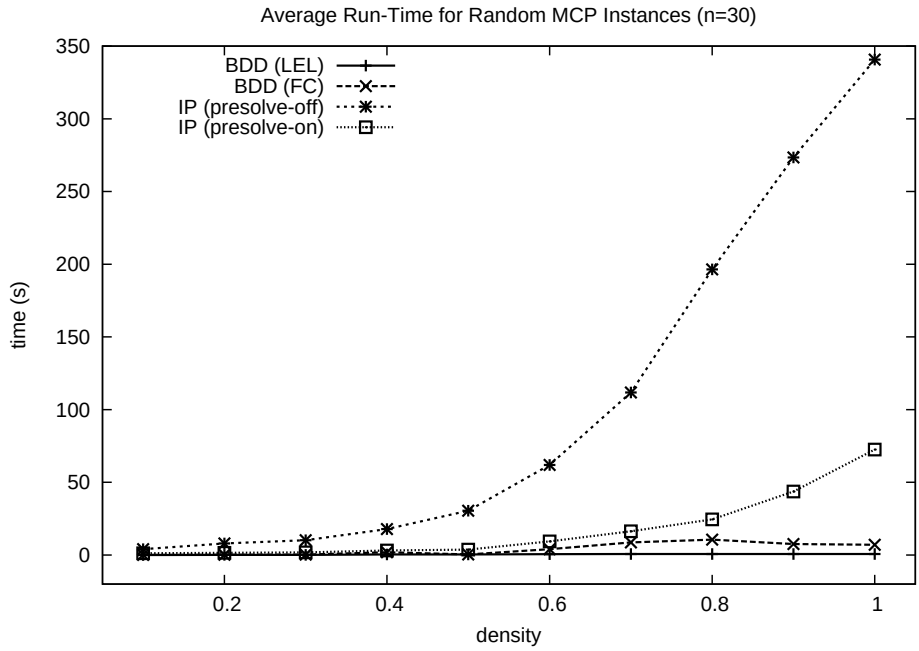

Figure 6: Average solution time for MCP instances ($n = 30$ vertices) using BDDs (with LEL and FC cutsets) and CPLEX (with and without presolve). Each point is the average of 10 random instances.

exponentially. Our main concern is whether the choice of state variables provides good bounds, given that we merge states that lie on poor solution paths. This opens the door to practical use of recursive models that have been historically intractable due to the curse of dimensionality.

We tested the BDD solver on three standard problems: the stable set problem mentioned earlier, the maximum cut problem, and the maximum 2-SAT problem. We deliberately chose problems that have simple and well-accepted MIP models. We found that the BDD solver is competitive with or superior to a state-of-the-art commercial MIP solver on these instances.

Figure 6 shows results for the maximum cut problem, in which a graph with edge weights is given, and the objective is to partition the vertices into two sets connected by edges of maximum total weight. Obviously, the MIP solver benefits greatly from the presolve routine, but even with it, the BDD-based solver is faster on all but the sparsest instances, and much faster on the denser instances. In fact, the solution time is indistinguishable from zero on the graph for all instances. Figure 7 shows performance profiles for the maximum 2-SAT problem using two types of branching procedures in the relaxed BDD. Here the objective is to satisfy the maximum number of logical clauses in a 2-satisfiability problem. MIP is faster at solving the easier instances (where a fast solver is not needed anyway), but the BDD solver excels on the harder instances.

The real test of a BDD-based solver is on the many problems that have dynamic programming models but no convenient MIP model. Even problems with simple MIP models may become difficult to formulate in MIP when side constraints are added, whereas a dynamic programing model is very flexible at accommodating side constraints and state-dependent objective functions. The constraints and objective function need only be expressible in terms of the current state and control. Black-box constraints are easily incorporated and actually make the problem easier by reducing the size of the

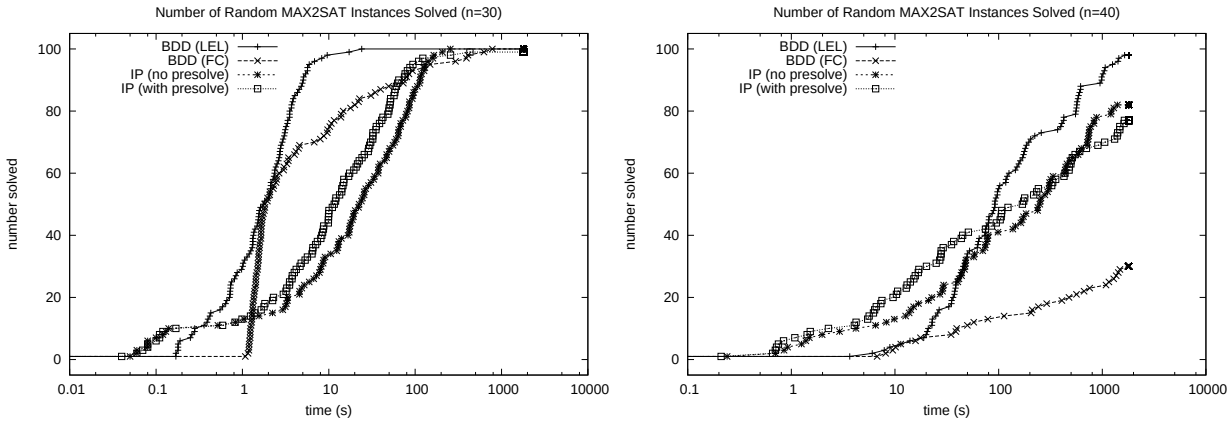

Figure 7: Time profile for 100 MAX-2SAT instances with $n = 30$ variables (left) and $n = 40$ variables (right), comparing BDDs and CPLEX (with and without presolve). LEL and FC refer to two branching schemes in the relaxed BDD.

relaxed BDD.

As future research, we intend to address large industrial problems with recursive formulations that are intractable for conventional dynamic programming and difficult to model as MIPs. In fact, BDD-based optimization has the potential to scale up to sizes that are beyond the reach of MIP, because there is no need to load a large inequality model into the solver. The model is essentially constant size because it is a recursion. The largest data structure is the relaxed BDD, whose size is controlled by the width. The results reported above were obtained with a maximum width of only 5 at nodes below the root node, so that memory requirements were not an issue. Recent experience suggests that a BDD-based solver also parallelizes much more efficiently than MIP, with near-linear speedups. These factors point the way toward successful large-scale optimization with BDD-based solvers that use dynamic programming models.

6 Logic-Based Benders Decomposition

Logic-based Benders decomposition (LBBD) has been used for some years to combine CP and MIP, usually by solving the master problem with MIP and the subproblem with CP. LBBD has the advantage that the subproblem need not be a linear (or nonlinear) inequality model as in classical Benders decomposition, but can be any optimization problem. Benders cuts are generated by solving the inference dual of the subproblem, which is analogous to the linear programming dual in classical Benders.

We advanced LBBD on four fronts. First, we did a fresh comparison in [13] of LBBD with state-of-the-art MIP on planning and scheduling problems, in response to some reports that LBBD was losing its advantage due to advances in MIP. We found that LBBD continues to run 100 to 1000 times faster than the latest commercial MIP solvers on these problems. This is partly because LBBD improves when MIP improves, due to the fact that the master problem is solved by MIP.

Second, we applied LBBD in [14] to a problem that does not decompose naturally, namely a single-facility scheduling problem with a long time horizon. We created subproblems by breaking the time horizon into segments, which requires rather complicated Benders cuts. Some sample results

Table 1: Computation times in seconds for the segmented problem with tight time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

	Feasibility			Makespan			Tardiness		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.1	14	1.9	60	7.7	6.4	0.1	16	3.0
80	181*	45	2.7	420*	147	11	63*	471*	20
100	199*	58	4.3	600*	600	17	547*	177*	11
120	272*	137	4.8	600*	600	39	600*	217*	2.9
140	306*	260*	6.8	600*	432*†	33	600*	373*	5.0
160	314*	301*	8.0	600*	359*	14			
180	600*	350*†	4.8	600*	557*†	5.3			
200	600*	†	5.8	600*	600*†	6.6			

*Solution terminated at 600 seconds for some or all instances.
†MILP solver ran out of memory for some or all instances, which are omitted from the average solution time.

appear in Table 1 for three types of problems: feasibility problems, minimum makespan problems, and minimum tardiness problems. In these instances, the processing of a job cannot overlap two adjacent time segments, as when the shop shuts down for weekends. LBBD is clearly superior to MIP and CP. The advantage of LBBD is less dramatic when jobs can overlap adjacent segments, but the advantage remains, and it is likely to increase as the time horizon grows. This is because the MIP model relies on time-indexed variables that increase in number for longer time horizons, while the time segments and therefore the LBBD subproblems can remain constant size.

Third, we applied LBBD to robust scheduling in a IT service center scheduling problem, and a preliminary report appears in the conference paper [15]. We use robust scheduling with an empirically determined uncertainty set. Vectors of processing delays for the jobs are assumed to lie in a polyhedral uncertainty set that represents realistic contingencies. An optimal robust schedule is one that minimizes the worst-case total delay, where the worst case is defined as the worst case in the uncertainty set. We prove that an optimal schedule can be obtained by examining only extreme points of the uncertainty set. Furthermore, we show that this problem can be formulated using an LBBD model in which the worst-case calculation is the subproblem. This allows the use of Benders cuts to speed solution, which has not been previously done in robust optimization. Computational testing is still underway, but preliminary results indicate that LBBD is faster than MIP.

Finally, we combined LBBD with decision diagrams by representing the Benders master problem as a decision diagram. When a Benders cut is generated, they are reflected in the master problem by modifying the decision diagram in a way that excludes the same solutions excluded by the Benders cuts. This poses a general separation problem for decision diagrams, which is analogous to the separation problem for polyhedral theory. We prove in [16] that the separating decision diagram can grow exponentially as cuts are added, but in practice it tends to grow linearly (Fig. 8). Our goal is to apply this technique to the home health care scheduling problem, work that is now underway.

7 Unifying Exact and Heuristic Methods

We argue in [17] that many exact and heuristic methods have common structure that permits some degree of unification. This is because many solution algorithms can be interpreted as primal-dual methods in which the primal component searches over problem restrictions, and the dual component obtains bounds on the optimal value. In particular, the concept of an inference dual provides the

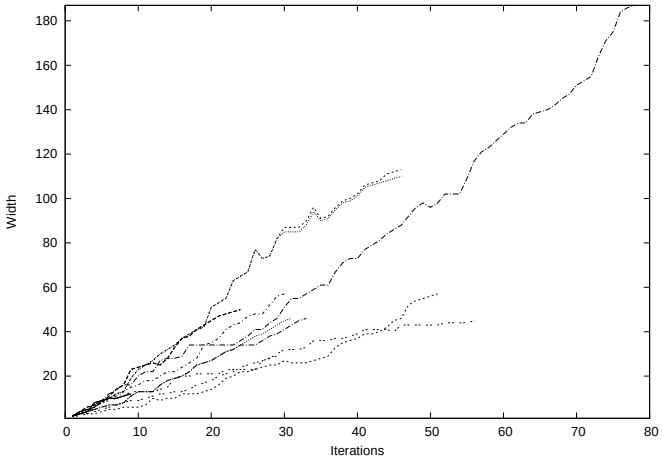

Figure 8: Growth of the separating decision diagram in the master problem, versus the number of Benders iterations, for several instances of a home health care scheduling problem. The curves terminate when the instance is solved.

basis for constraint-directed search, which is a feature of many exact and heuristic methods. The motivations for unification are (a) to encourage the exchange of algorithmic techniques between exact and heuristic methods, and (b) to design solution methods that transition gracefully from exact to heuristic modes as problem instances scale up.

We first identify primal-dual structure various exact algorithms, as summarized in Table 2. These include the simplex method, branch-and-bound-methods, satisfiability solvers, and Benders decomposition. We then find the same structure in such heuristic methods as local search, tabu search, evolutionary algorithms, ant colony optimization, and particle swarm optimization. The scheme appears in Table 3. We also indicate how algorithmic ideas can be exchanged between exact and heuristic methods, and where possible, how a single algorithm can be designed to transition from one to the other.

References

[1] J. N. Hooker, Hybrid modeling, in M. Milano and P. Van Hentenryck, eds., *Hybrid Optimization: The Ten Years of CPAIOR*, Springer (2011) 11–62.

[2] A. Ciré, J. N. Hooker, and T. H. Yunes, Modeling with metaconstraints and semantic typing of variables, submitted, 2014.

[3] D. Bergman and J. N. Hooker, Graph coloring facets from all-different systems, in N. Beldiceanu, N. Jussien and E. Pinson, eds., *CPAIOR 2012 Proceedings*, 50-65.

[4] D. Bergman and J. N. Hooker, Graph coloring inequalities from all-different systems, *Constraints* **19** (2014), 404-433.

[5] D. Bergman, W.-J. Van Hoeve and J. N. Hooker, Manipulating MDD relaxations for combinatorial optimization, in T. Achterberg and J. S. Beck, eds., *CPAIOR 2011 Proceedings*, 20–35.

Table 2: Summary of exact methods.

Method	*Restrictions Enumerated*	*Relaxation Dual Providing Bounds*	*Nogoods from Inference Dual*	*To Convert to a Heuristic Method*
DPLL (for SAT)	Leaf nodes of search tree		Conflict clauses	Drop some conflict clauses
Simplex (for LP)	Edges of polyhedron	Surrogate dual	Reduced costs	Terminate prematurely
Branch & bound (for IP)	Leaf nodes of search tree	LP bounds	Conflict analysis	Forget tree and use nogood constraints
Branch & bound as local search	Incomplete search trees	Uphill search over incomplete trees		Terminate prematurely
Benders decomposition	Subproblems	Master problem	Logic-based Benders cuts	Drop some Benders cuts

Table 3: Summary of heuristic methods.

Method	*Restrictions Enumerated*	*Relaxation Dual Providing Bounds*	*Nogoods from Inference Dual*	*To Convert to an Exact Method*
Local search (e.g., for TSP)	Neighborhoods (e.g., 2-opt nbhds)	Adjustment of neighborhood size		Search over partial solutions
GRASP (e.g., for TSPTW)	Nodes of incomplete search tree	As in branch & bound		Don't forget nodes, complete the tree
Tabu search (e.g., for TSPTW)	Neighborhoods		Items on tabu list	Search over partial solutions; dynamic backtracking
Genetic algorithm	Populations		Crossover guidance from inductive inference dual	
Genetic algorithm as local search	Subsets of population plus offspring	Control of neighborhood size	Logic-based	
Ant colony optimization	Same as GRASP	As in GRASP		As in GRASP
Particle swarm optimization	Sets of swarm locations		Relocation guidance from inductive inference dual	

[6] D. Bergman, A. Ciré, W.-J. van Hoeve, and J. N. Hooker, Variable ordering for the application of BDDs to the maximum independent set problem, in N. Beldiceanu, N. Jussien and E. Pinson, eds., *CPAIOR 2012 Proceedings*, 34–49.

[7] D. Bergman, A. Ciré, W.-J. van Hoeve, and J. N. Hooker, Optimization bounds from binary decision diagrams, *INFORMS Journal on Computing* **26** (2013), 253–268.

[8] D. Bergman, A. Ciré, W.-J. van Hoeve, and T. Yunes, BDD-based heuristics for binary optimization, *Journal of Heuristics* **20** (2014), 211–234.

[9] D. Bergman, A. Ciré, W.-J. van Hoeve, and J. N. Hooker, Discrete optimization with decision diagrams, submitted 2014.

[10] J. N. Hooker, Decision diagrams and dynamic programming, C. Gomes and M. Sellmann, eds., *CPAIOR 2013 Proceedings*, 94–110.

[11] D. Bergman, New methods for discrete optimization, Ph.D. thesis, Carnegie Mellon University, 2013.

[12] A. Ciré, Decision diagrams for optimization, Ph.D. thesis, Carnegie Mellon University, 2014.

[13] A. Ciré, E. Çoban and J. N. Hooker, Mixed integer programming vs logic-based Benders decomposition for planning and scheduling, in C. Gomes and M. Sellmann, eds., *CPAIOR 2013 Proceedings*, 325-331.

[14] E. Çoban and J. N. Hooker, Single facility scheduling by logic-based Benders decomposition, *Annals of Operations Research* **210** (2013) 245–272.

[15] E. Çoban, A. Heching, J. N. Hooker, and A. Scheller-Wolf, Robust optimization with logic-based Benders decomposition, submitted, 2014

[16] A. A. Ciré and J. N. Hooker, The separation problem for binary decision diagrams, *ISAIM 2014 Proceedings*.

[17] J. N. Hooker, Toward unification of exact and heuristic optimization methods, *International Transactions in OR*, published online May 2013.