

Generalized Conflict Learning For Hybrid Discrete Linear Optimization *

Hui Li and Brian C. Williams

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{huili, williams}@mit.edu

Abstract

An increasing range of problems in Artificial Intelligence and Computer Science, such as autonomous vehicle control and planning with resources, are formulated through a combination of logical, algebraic and cost constraints. Their solution requires a hybrid mixture of logical decision techniques and mathematical optimization. Using Disjunctive Linear Programs (DLPs) to formulate these problems, we present a novel algorithm, called Generalized Conflict-Directed Branch and Bound (GCD-BB), that efficiently solves DLPs through a powerful three-fold method. First, during the search process, generalized conflict learning learns *conflicts* for regions of the state space that are infeasible or sub-optimal. Second, forward conflict-directed search uses these conflicts to guide the forward step of search, by moving away from regions of state space corresponding to known conflicts. Finally, induced unit clause relaxation automatically forms a strong relaxed problem from a subset of the unit clauses that are implied by the original problem. Our experiments on model-based temporal plan execution for cooperative vehicles demonstrate an order of magnitude speed-up over Binary Integer Programming Branch and Bound.

Introduction

An increasing range of problems in Artificial Intelligence and Computer Science involve finding optimal solutions to problems that involve a rich combination of logical and algebraic constraints, and require a hybrid coupling of logical decision techniques with mathematical optimization to solve. Examples include planning with resources, autonomous vehicle control and verification of timed and hybrid systems. Focusing on the area of autonomous vehicle control, deep space explorers must choose between tasks and temporal orderings, while optimizing flight trajectories for fuel usage. On Earth, search and rescue units must construct and compare different vehicle trajectories around dangerous areas, such as a fire, on the approach to a trapped individual.

Each of these tasks involves designing an optimal trajectory, based on a continuous dynamic model. At some point, they must satisfy additional logical constraints, such as mission tasks, task orderings and obstacle avoidance.

*This research is funded by The Boeing Company grant MIT-BA-GTA-1 and by NASA grant NNA04CK91A.

These Hybrid Discrete/ Linear Optimization Problems (HDLOPs) can be formulated in three ways: first, by introducing integer variables and corresponding constraints to Linear Programming (LP), known as Binary Integer Programming (BIP) (Schouwenaars *et al.* 2001)(Vossen *et al.* 1999)(Kautz & Walser 1999); second, by augmenting LP with propositional variables so that the propositional variables can be used to trigger linear constraints, known as Mixed Logic Linear Programming (MLLP) (Hooker & Osorio 1999) (although MLLP is a generalization from BIP, its main feature is the introduction of propositional variables.), and LCNF (Wolfman & Weld 1999) (note that optimization is not involved in LCNF.); third, by extending LP with disjunctions, without adding any variables or constraints, called Disjunctive Linear Programming (Balas 1979). This paper builds upon the last option, which combines the expressive power of propositional logic with that of LP, without the overhead of additional variables or constraints.

In this paper we introduce a novel algorithm for efficiently solving Disjunctive Linear Programs (DLPs) called Generalized Conflict-Directed Branch and Bound (GCD-BB). It extends the Branch and Bound (B&B) algorithm by using logical inference to do relaxation, by extracting the descriptions of the source of discovered infeasibility and sub-optimality as conflicts to guide the forward search, so as to prune the state space. Our experiments, comparing GCD-BB against Binary Integer Programming B&B (BIP-BB), demonstrate a significant performance gain on model-based temporal plan execution for cooperative vehicles.

Problem Formulation

We use DLPs to effectively capture both the continuous dynamics and control decisions present in HDLOPs. Fig.1 depicts a simple example of an HDLOP, introduced in (Schouwenaars *et al.* 2001). In particular, a vehicle has to go from point A to C, without hitting the obstacle B, while minimizing fuel use. Eq.1 describes its DLP formulation.

$$\begin{aligned} & \text{Minimize } f(x) \\ & \text{Subject to } g(x) \leq 0 \\ & x_t \leq x_L \vee x_t \geq x_R \vee y_t \leq y_B \vee y_t \geq y_T, \\ & \forall t = 1, \dots, n \end{aligned} \tag{1}$$

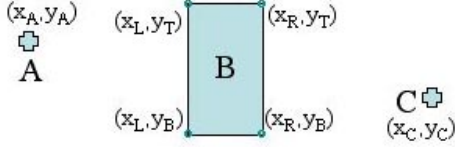


Figure 1: A simple example of an HDLOP

In Eq.1, \vee denotes logical *or*, x is a vector of decision variables that includes, at each time step t , the position, velocity and acceleration of the vehicle; $f(x)$ is a linear cost function in terms of fuel use; $g(x) \leq 0$ is a conjunction of linear constraints on vehicle dynamics, and the last constraint keeps the vehicle outside obstacle B at each time step t .

$$\begin{aligned} & \text{Minimize } f(x) \\ & \text{Subject to } \bigwedge_{i=1, \dots, n} \left(\bigvee_{j=1, \dots, m_i} C_{ij}(x) \leq 0 \right) \quad (2) \end{aligned}$$

In general, a DLP takes the form shown in Eq.2, where x is a vector of decision variables, $f(x)$ is a linear cost function, and the constraints are a conjunction of n clauses, each of which (clause i) is a disjunction of m_i linear inequalities, $C_{ij}(x) \leq 0$. A DLP reduces to a standard LP in the special case when $m_i = 1, \forall i = 1, \dots, n$.

The GCD-BB Algorithm

The GCD-BB algorithm has three key features: first, Generalized Conflict Learning learns *conflicts* comprised of constraint sets that produce either infeasibility or sub-optimality; second, Forward Conflict-Directed Search guides the forward step of search away from regions of state space corresponding to known *conflicts*; third, Induced Unit Clause Relaxation forms a relaxed problem from a subset of the unit clauses that are induced from the original problem. Finally, this paper also explores the impact of search order: best-first search (BFS) versus depth-first search (DFS). In the following subsections, we develop these key features in detail, including examples and pseudo code.

B&B For DLPs

GCD-BB builds upon B&B, which is frequently used by BIP, to solve problems involving both discrete and continuous variables. Instead of exploring the entire feasible set of a constrained problem, B&B uses bounds on the optimal cost, in order to avoid exploring subsets of the feasible set that it can prove are sub-optimal, that is, subsets whose optimal solution is not better than the *incumbent*, which is the best solution found so far.

The search tree of B&B for BIP branches by assigning values to the integer variables. In our case, B&B for DLP branches by splitting clauses: that is, a tree node is expanded by selecting one of the DLP clauses, and then selecting one of the disjuncts of the clause for each of the child nodes. At each node in the search tree, a relaxed LP is solved. p' is a relaxed LP of an optimization problem p , if the feasible region of p' contains the feasible region of p , and they have

the same objective function. Therefore if p' is infeasible, then p is infeasible. Assuming minimization, if p' is solved with an optimal value v , the optimal value of p is guaranteed to be greater than or equal to v . B&B uses relaxed problems to obtain lower bounds of the original problem.

Alg. 1 BB-DLP(DLP)

```

1:  $upperBound \leftarrow +\infty$ 
2:  $timestamp = 0$ 
3: put  $DLP$  into a FILO queue
4: while queue is not empty do
5:    $node \leftarrow$  remove from queue
6:    $node.relaxedSolution \leftarrow$  solveLP( $node.relaxedLP$ )
7:   if  $node.relaxedLP$  is infeasible then
8:     continue { $node$  is deleted}
9:   else if  $node.relaxedValue \geq upperBound$  then
10:    continue { $node$  is deleted}
11:  else
12:     $expand = \text{False}$ 
13:    for each clause in  $node.nonUnitClauses$  do
14:      if Violated-Clause?( $clause,$ 
15:         $node.relaxedSolution$ ) then
16:         $expand \leftarrow \text{True}$ 
17:        break
18:      end if
19:    end for
20:    if  $expand = \text{False}$  then
21:       $upperBound \leftarrow node.relaxedValue$  { $a$  new
22:         $incumbent$  was found}
23:       $incumbent \leftarrow node.relaxedSolution$ 
24:    else
25:      put Expand-Node( $node, timestamp$ ) in queue
26:       $timestamp \leftarrow timestamp + 1$ 
27:    end if
28:  end while
29: if  $upperBound < +\infty$  then
30:   return  $incumbent$ 
31: else
32:   return INFEASIBLE
33: end if

```

Alg. 2 Violated-Clause?($clause, solution$)

```

1: for each disjunct in  $clause$  do
2:   if  $solution$  satisfies  $disjunct$  then
3:     return False
4:   end if
5: end for
6: return True

```

Pseudo code for B&B for DLPs is shown in Alg.1. If a node is feasible (line 9) and better than the incumbent (line 11), it is tested (line 13-18) whether further expansion is needed, using Violated-Clause? (Alg.2). For example, if the node does not need expansion (line 19), it is identified as the new incumbent; otherwise, Expand-Node (Alg.3) is called,

which creates the children of the node and places them in the queue (line 23).

Alg. 3 Expand-Node(*node*, *timestamp*)

```

1: for each clause in node.nonUnitClauses do
2:   if Violated-Clause?(clause, node.relaxedSolution)
3:     then
4:       add clause to sortList {sortList contains violated
5:         clauses in increasing order of their number of
6:         disjuncts}
7:   end if
8: end for
9: selectedClause ← sortList(first)
10: for each disjunct in selectedClause do
11:   child.unitClauses ← node.unitClauses + disjunct
12:   child.nonUnitClauses ← node.nonUnitClauses -
13:     selectedClause
14:   child.timestamp ← timestamp + 1
15:   add child to childList
16: end for
17: return childList

```

B&B for DLPs differs from B&B for BIPs in three respects. First, it forms relaxed LPs from unit clauses, rather than extending the domain of binary variables to the real-valued interval $[0,1]$. Second, the condition for performing node expansion is the existence of violated clauses, rather than unsatisfied (real-valued) binary variables. Finally, it expands nodes by splitting clauses, rather than assigning binary variables 0 and 1.

Generalized Conflict Learning

In the related field of discrete constraint satisfaction, conflict-directed methods, such as dependency-directed backtracking (Stallman & Sussman 1977), backjumping (Gaschnig 1978), conflict-directed backjumping (Prosser 1993) and dynamic backtracking (Ginsberg 1993), dramatically improve the performance of backtrack (BT) search, by learning the source of each inconsistency discovered, and by using this generalization, called a *conflict* (or a *nogood*), to prune additional subtrees that the conflict identifies as inconsistent.

To apply conflict learning to B&B, we note that B&B prunes subtrees corresponding to relaxed subproblems that are sub-optimal and infeasible. Hence two opportunities exist for learning and pruning. We generalize conflict learning and pruning, in contrast to previous work, in that conflicts are extracted from both sub-optimal and infeasible subproblems. Moreover, it is valuable to have the description as compact as possible, so that the subspace that can be pruned is as large as possible.

In the context of DLP, each *conflict* can be one of two types: an *infeasibility conflict*, or a *sub-optimality conflict*. An *infeasibility conflict* is a set of inconsistent constraints of an infeasible subproblem. An example is the constraint set $\{a,b,c,d\}$ in Fig.2(a). A *sub-optimality conflict* is a set of active constraints of a suboptimal subproblem. An inequality constraint $g_i(x) \leq 0$ is *active* at a feasible point \tilde{x}

if $g_i(\tilde{x}) = 0$. An example of a sub-optimality conflict is the constraint set $\{a,b,d\}$ in Fig.2(b).

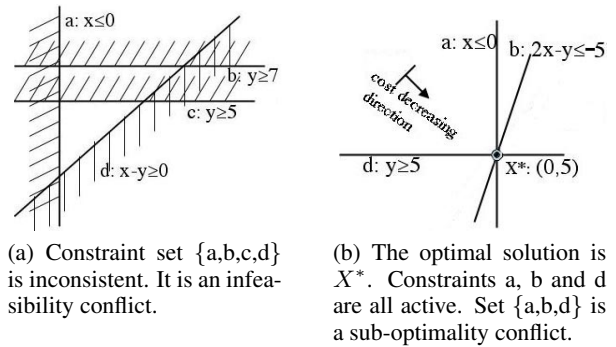


Figure 2: Conflicts

In order to prune a larger portion of the state space, we extract *minimal conflicts* rather than any conflicts. A set of constraints is *minimal* if removing any one of the constraints from the set resolves the infeasibility or sub-optimality. For example, the constraint set $\{a,c,d\}$ in Fig.3(a) is a minimal conflict, as it is an inconsistent constraint set and every proper subset of it is consistent. Constraint set $\{a,d\}$ in Fig.3(b) is also a minimal conflict. Note that there can be more than one minimal set (possibly with different cardinalities) involved in one infeasibility or sub-optimality, and a minimal conflict is not guaranteed to have the minimum cardinality.

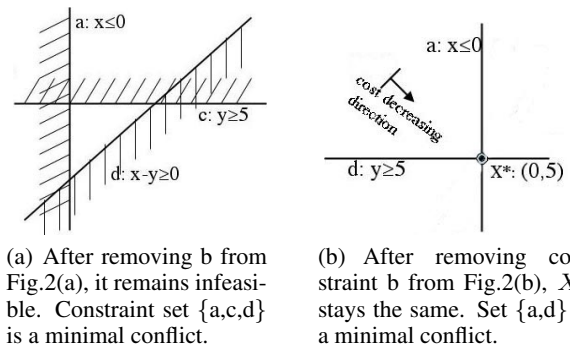


Figure 3: Minimal Conflicts

To perform generalized conflict learning efficiently, we introduce two novel methods based on the duality theory (Bertsimas & Tsitsiklis 1997) to extract a subproblem's minimal conflict. Recall that in the Branch and Bound search tree, a relaxed LP is solved at each node. We run the dual simplex method as the LP solver. For infeasibility, when dual simplex terminates unboundedly, an extreme ray is discovered with it. The non-zero elements of the extreme ray are used to identify the constraints of the minimal infeasibility conflict. The proof is in the working paper (Li 2005b). For sub-optimality, when dual simplex terminates with an optimal solution, we can choose to extract the minimal sub-

optimality conflict. To accomplish that, we reduce the constraint matrix so that there are no duplicate constraints and examine the consequent dual solution vector. Suppose the reduced constraint matrix has m rows and n columns. Then the corresponding dual vector has m elements. If there are n or less non-zero elements in the dual vector, all the non-zero elements of the dual vector correspond to the constraints of the minimal sub-optimality conflict; otherwise, *any* n of the non-zero elements are used to identify the minimal conflict. The principle is explained in (Li 2005a) ,

Forward Conflict-directed Search

We use forward conflict-directed search to heuristically guide the forward step of search away from regions of the state space that are ruled out by known conflicts. Backward search methods also use conflicts to direct search, such as dependency-directed backtracking (Stallman & Sussman 1977), backjumping (Gaschnig 1978), conflict-directed backjumping (Prosser 1993), dynamic backtracking (Ginsberg 1993) and LPSAT (Wolfman & Weld 1999). These backtrack search methods use conflicts to select backtrack points and as a cache to prune nodes without testing consistency. We use conflicts in forward search, as in conflict-directed A* (CD-A*) search (Williams & Ragno 2005), to move away from known “bad” states. We generalize the approach in CD-A* to guiding B&B away from regions of state space that the known conflicts indicate are infeasible or sub-optimal. To accomplish this, a node is expanded so that each of its children resolves all the node’s unresolved conflicts. A node *resolves* a conflict if at least one of the conflict’s disjuncts is explicitly excluded from the relaxed LP of the node.

In terms of implementation, we replace function Expand-Node in line 24 of BB-DLP (Alg. 1) with function General-Expand (Alg. 4). When there is no unresolved conflict, Expand-Node (Alg. 3) is used, and when unresolved conflicts exist, Forward-CD-Search is performed.

Alg. 4 General-Expand(*node, timestamp, conflictDB*)

```

1: conflictSet ← conflictDB(timestamp)
2: if conflictSet is empty then
3:   Expand-Node(node, timestamp)
4: else
5:   Forward-CD-Search(node, conflictSet)
6: end if

```

Forward-CD-Search includes three steps: 1) Generate-Constituent-Kernels, 2) Generate-Kernels and 3) Generate-And-Test-DLP-Candidates. Pseudo code of the functions can be found in (Li 2005a). An example of the three steps is shown in Fig. 4, where each linear inequality is represented by a symbol for convenience.

A *constituent kernel* is a minimal description of the states that resolve a conflict. In the context of DLPs, a constituent kernel of a conflict is a linear inequality that is the negation of a linear constraint contained in the conflict. Once the constituent kernels for all the conflicts are generated, we use minimal set covering to generate the kernels. Finally, a DLP candidate is generated for each kernel, as shown in Fig. 4,

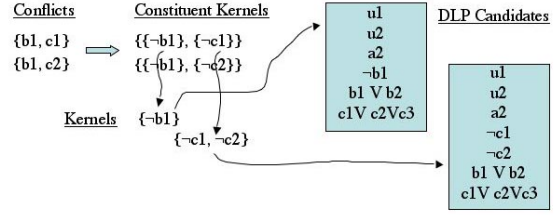


Figure 4: Each conflict is mapped to a set of constituent kernels, which resolve that conflict alone. Kernels are generated by combining the constituent kernels using minimal set covering. A DLP candidate is formed for each kernel, and is checked for consistency.

and the propositionally unsatisfiable DLPs are pruned, using a fast unit propagation test before solving any relaxed LP.

Our experimental results on a range of cooperative vehicle plan execution problems show that forward conflict-directed search significantly outperforms backtrack search with conflicts.

Induced Unit Clause Relaxation

Relaxation provides bounds on feasibility and the optimal value of a problem, which are commonly used by B&B to prune the search space. Previous research (Hooker 2002) typically solves DLPs by reformulating them as BIPs, where a relaxed LP is formed by relaxing the binary constraints ($x \in \{0, 1\}$) to a corresponding set of continuous linear constraints ($0 \leq x \leq 1$).

An alternative way of creating a relaxed LP is to operate on the DLP encoding directly, by removing all non-unit clauses from the DLP (a unit clause is one that contains a single linear inequality). The rationale in (Hooker 2002) for reformulating a DLP as a BIP relaxation, is that it maintains some of the constraints of the non-unit clauses through the continuous relaxation from binary to real-valued variables; this is opposed to ignoring all the non-unit clauses. However, this benefit is at the cost of adding binary variables and constraints, which increases the dimensionality of the search problem.

Our approach leverages the reduced state space, by starting with the direct DLP relaxation. We overcome the weakness of standard DLP relaxation (loss of non-unit clauses) by adding to the relaxation, unit clauses that are logically entailed by the original DLP.

This relaxation is defined by Induce-Unit-Clause (Alg. 5) and demonstrated in Fig. 5. In the experiment section we compare our induced unit clause relaxation with the BIP relaxation and show a profound improvement on a range of cooperative vehicle plan execution problems.

Induce-Unit-Clause (Alg. 5) performs unit propagation among the unit and non-unit clauses to induce more unit clauses.

Induce-Unit-Clause simplifies a DLP by performing unit propagation on the propositional clause set of the DLP, and then reflecting the consequences of the propagation back

Alg. 5 Induce-Unit-Clause(DLP)

```
1:  $\{DLP.unitClauses, DLP.nonUnitClauses\} \leftarrow$   
   Unit-Propagation( $\{DLP.unitClauses,$   
    $DLP.nonUnitClauses\}$ )  
2:  $DLP.relaxedLP \leftarrow < DLP.objective,$   
    $DLP.unitClauses >$   
3: return  $DLP$ 
```

on the original DLP. More specifically, in line 1 of Alg. 5, Induce-Unit-Clause calls function Unit-Propagation to simplify the unit and non-unit clause sets of a DLP (Steps (2)-(5) in Fig. 5). A relaxed LP is also formed by combining the objective function and the unit clause set (line 2).

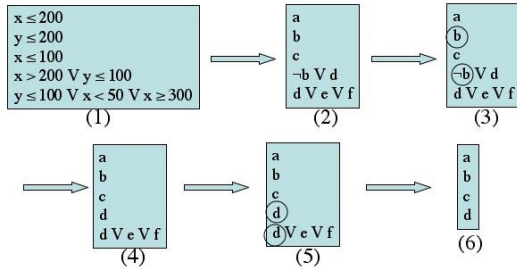


Figure 5: An example of induced unit clause relaxation

Search Order: Best-first versus Depth-first

Given a fixed set of heuristic information, (Dechter & Pearl 1985) shows that best-first search is the most efficient algorithm in terms of time efficiency. Intuitively, this is because BFS does not visit any node whose heuristic value is worse than the optimum, and all nodes better than the optimum must be visited to ensure that the optimum is not missed. However, BFS can take dramatically more memory space than DFS. Nevertheless, with conflict learning and forward conflict-directed search, the queue of the BFS search tree is significantly reduced. Our experimental results show that BFS can take memory space similar to DFS, while taking less time to find the optimum.

An additional issue for GCD-BB is that the concept of sub-optimality is rooted in maintaining an incumbent. Hence, it can be applied to DFS but not to BFS. To evaluate these tradeoffs, our experiments in the next section compare the use of BFS and conflict learning from infeasibility only, with DFS and conflict learning from both infeasibility and from sub-optimality.

Experimental Performance Analysis

This section provides experimental results of the GCD-BB algorithm on a range of test problems, for coordinated air vehicle control (Léauté & Williams 2005). GCD-BB is compared with the benchmark B&B algorithm applied both to DLPs and their equivalent BIP encoding. We also compare the effect of several algorithmic variants, in particular, infeasibility conflict learning versus sub-optimality con-

flict learning, forward conflict-directed search versus backtrack conflict-directed search, and BFS versus DFS. While each algorithm terminates with the same optimal solution, GCD-BB achieves an order of magnitude speed-up over BIP-BB. Our experiments also show that the elements of the algorithm that are most important with respect to achieving this performance are: generalized conflict learning, forward conflict-directed search and induced unit clause relaxation.

As the bulk of the computational effort expended by HD-LOP B&B algorithms is devoted to solving relaxed LP problems, the total number and average size of these LPs are representative of the total computational effort involved in solving the HDLOPs. Note that extracting infeasibility conflicts and sub-optimality conflicts can be achieved as by-products of solving the LPs and, therefore, does not incur any additional LPs to solve. We use the total number of relaxed LPs solved and the average LP size as our LP solver and hardware independent measures of computation time. To measure memory space use, maximum queue size is used.

Our experiments show that the average size of LPs solved for BIPs is larger than that of the LPs solved for the DLP encoding, and that the average size of LPs solved by each DLP algorithm variant is similar to one another. The data table is not listed due to space limit of the paper.

Clause/ Variable	80/ 36	700/ 144	1492/ 300	2456/ 480	
BIP-BB	31.5	2009	4890	8133	
DLP BFS	<i>w.o.CL</i>	24.3	735.6	1569	2651
	<i>Inf</i>	19.2	67.3	96.3	130.2
	<i>BT</i>	23.1	396.7	887.8	1406
DLP DFS	<i>w.o.CL</i>	28.0	2014	3023	4662
	<i>Inf</i>	22.5	106.0	225.4	370.5
	<i>BT</i>	25.9	596.9	1260	1994
	<i>Sub+Inf</i>	22.1	76.4	84.4	102.9
	<i>Sub</i>	25.8	127.6	363.7	715.0

Table 1: Comparison on the number of relaxed LPs

We programmed BIP-BB, GCD-BB and its variations in Java. Test problems were generated using a model-based temporal planner (Léauté & Williams 2005), performing multi-vehicle search and rescue missions. For each Clause/Variable set, 15 problems were generated and the average was recorded in the tables. In the two tables, each column represents a set of test problems with similar dimensionality. For example, the first column 80/36 represents a set of test problems that all have 80 clauses and 36 variables. Each row specifies performance for a particular algorithm, in terms of the number of relaxed LPs solved by each algorithm or the maximum queue size of the search tree of each algorithm.

In Table 1, the number of relaxed LPs solved for each approach is recorded. The second row shows that BIP-BB solves more LPs than any other approach. The difference increases dramatically as the problem grows larger. The next three rows are dedicated to DLP with BFS. The addition of infeasibility conflict learning (Inf) significantly outperforms without conflict learning (w.o. CL). The method using conflict-directed backtrack search (BT), which uses infeasibility conflicts to check consistency of a relaxed LP before

solving it, performs dramatically worse than the method using forward conflict-directed search (Inf). The last five rows represent the variations of DLP with DFS. Within these five rows, the method that solves the least relaxed LPs is the method with both infeasibility and sub-optimality conflict learning (Sub+Inf). The worst case is w.o. CL.

Consider BFS versus DFS, using only infeasibility conflict learning, BFS performs better than DFS, but the performance of DFS with Sub+Inf is close to BFS with Inf. For very large problems, DFS with Sub+Inf performs better. Under the same situation, either w.o. CL or with Inf or with BT, BFS solves less relaxed LPs than DFS, as explained before in Section *Search Order: Best-first versus Depth-first*. As the only difference between DLP with DFS without conflict learning and BIP-BB is in the formulation and relaxation methods, the significant improvement of the former over the latter verifies the statement in Section *Induced Unit Clause Relaxation*.

For all tests, our GCD-BB algorithm, using either DLP with BFS and infeasibility conflict learning, or DLP with DFS and infeasibility plus sub-optimality conflict learning, performs the best and has a profound improvement over BIP-BB on large problems.

In Table 2, all approaches have similar maximum queue sizes, except BFS w.o. CL and BFS with BT. As discussed earlier, BFS generally takes more memory space than DFS, but when forward conflict-directed search is used, the search space is reduced and the corresponding queue is shortened. Note that although the methods using BT are conflict-directed, the queue size of the one with BFS is not largely reduced.

Clause/ Variable		80/ 36	700/ 144	1492/ 300	2456/ 480
BIP-BB		8.4	30.8	46.2	58.7
DLP BFS	<i>w.o.CL</i>	19.1	161.1	296.8	419.0
	<i>Inf</i>	6.4	18.3	38.4	52.5
	<i>BT</i>	15.6	101.7	205.1	327.8
DLP DFS	<i>w.o.CL</i>	6.1	18.7	25.1	30.3
	<i>Inf</i>	6.5	21.4	45.0	57.3
	<i>BT</i>	6.1	18.4	23.5	28.1
	<i>Sub+Inf</i>	6.5	21.4	33.0	40.9
	<i>Sub</i>	6.5	21.6	38.7	47.0

Table 2: Comparison on the maximum queue size

Conclusion

This paper presented a novel algorithm, Generalized Conflict-Directed Branch and Bound, that efficiently solves DLP problems through a powerful three-fold method, featuring *generalized conflict learning*, *forward conflict-directed search* and *induced unit clause relaxation*. The key feature of the approach reasons about infeasible or sub-optimal subsets of state space using conflicts, in order to guide the forward step of search, by moving away from regions of state space corresponding to known conflicts. Our experiments on model-based temporal plan execution for cooperative vehicles demonstrated an order of magnitude speed-up over BIP-BB.

References

- Balas, E. 1979. Disjunctive programming. *A. of Discrete Math.* 5.
- Bertsimas, D., and Tsitsiklis, J. 1997. *Introduction to Linear Optimization*. Athena Scientific.
- Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *ACM* 32.
- Gaschnig, J. 1978. Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisfying assignment problems. In *The 2nd Canadian Conference on AI*.
- Ginsberg, M. 1993. Dynamic backtracking. *J. of AIR* 1:25–46.
- Hooker, J., and Osorio, M. 1999. Mixed logical/linear programming. *J. of DAM* 96-97.
- Hooker, J. 2002. Logic, optimization and constraint programming. *INFORMS J. on Computing* 14.
- Kautz, H., and Walser, J. 1999. State space planning by integer optimization. In *AAAI*.
- Léauté, T., and Williams, B. 2005. Coordinating agile systems through the model-based execution of temporal plans. In *AAAI*.
- Li, H. 2005a. Generalized conflict learning for hybrid discrete/ linear optimization. Master’s thesis, MIT.
- Li, H. 2005b. On the “minimum” extreme ray of unbounded LPs. *Working paper*.
- Prosser, P. 1993. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9(3).
- Schouwenaars, T.; de Moor, B.; Feron, E.; and How, J. 2001. Mixed integer programming for multi-vehicle path planning. In *European Control Conference*.
- Stallman, R., and Sussman, G. 1977. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *J. of AI* 9.
- Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *IJCAI*.
- Williams, B., and Ragno, R. 2005. Conflict-directed A* and its role in model-based embedded systems. *To appear in J. of DAM*.
- Wolfman, S., and Weld, D. 1999. The Ipsat engine & its application to resource planning. In *IJCAI*.