BPSolver's Solutions to the Third ASP Competition Problems

Neng-Fa Zhou[1], Agostino Dovier[2], and Yuanlin Zhang[3]

[1] CUNY Brooklyn College & Graduate Center
[2] Univ. di Udine
[3] Texas Tech University

1 Introduction

Continuing with its participation in the second ASP competition, BPSolver participated in the third ASP competition. There were two tracks in the third competition, namely, the *System* track and the *Model & Solve* track. While in the *System* track, the participating solvers competed on solving benchmark problems uniformly encoded in the standard ASP, in the *Model & Solve* track, solvers were allowed to use their own encodings of the problems. BPSolver participated in the *Model & Solve* track with five other solvers including Clasp, AClasp, EZCSP, IDP, and Fastdownward. BPSolver and Fastdownward were two non-ASP solvers. BPSolver was completely based on B-Prolog, a standard top-down goal-driven Prolog system endowed with tabling and CLP(FD). Fastdownward was a PDDL compiler. Strictly speaking, IDP, a model generation system based on first-order logic, is not an ASP solver either, but it resembles ASP solvers in that it uses a bottom-up grounder to transform programs into propositional ones and uses a SAT solver to compute models. Some of the ASP-based solvers also used CP solvers for some of the benchmarks. For example, Clasp used Gecode and EZCSP used B-Prolog for the packing and scheduling benchmarks.

A total number of 34 benchmarks were used in the *Model & Solve* track. These problems were classified into four categories: P problems (7), NP problems (19), beyond NP problems (2), and optimization problems (6). The classification of some of the problems was debatable. For example, the *Stable Marriage* problem was classified as a P problem, but the preferences given in the problem instances were partially ordered, which made it impossible to use the famous Gale and Shapley algorithm [2]. Also, the *Hanoi Tower* problem was classified as an NP problem, but our dynamic programming solution clearly demonstrated the polynomial complexity of the problem in the number of moves.

BPSolver took the second place, preceded by Clasp, in the overall ranking. Another solver from Potsdam, AClasp, placed the third, followed by EZCSP. The solver Fastdownward, which only participated in the planning benchmarks, ranked last. If the solvers were ranked on the total number of benchmarks won, Clasp would remain at the top with 18 wins and BPSolver would still take the second place with 14 wins.

Except for four problems (*Company Control*, *Grammar*, *Labyrinth*, and *Tomography*) that were solved using only plain Prolog, all the BPSolver's solu-

Table 1. Benchmarks won by BPSolver

Score	Benchmark	Used Feature
100	Tangram*	CLP(FD)
100	Magic Square Sets	CLP(FD)
100	Hydraulic Planning*	Tabling
100	Hydraulic Leaking	Tabling
097	Weight-Assignment Tree	CLP(FD)
097	Knight Tour*	CLP(FD)
096	Grammar-Based IE	Prolog
094	Hanoi Tower**	Tabling
093	Airport Pickup	Tabling
092	Disjunctive Scheduling	CLP(FD)
090	Reachability	Tabling
072	Labyrinth	Prolog
039	Tomography	Prolog
037	Maximal Clique	CLP(FD)

tions used either CLP(FD) or tabling. Most of the high-performing CLP(FD) programs used global constraints including `all_distinct`, `element`, `circuit`, `cumulative`, and `path_from_to`. Mode-directed tabling demonstrated a strong performance in the competition. It not only helped easily solve the path-finding problems such as *Airport-Pickup*, *Hydraulic Planning*, and *Hydraulic Leaking* problems, but also helped provide elegant and efficient dynamic programming solutions to the *Sokoban* and *Hanoi Tower* problems for which neither tabling nor CLP(FD) had been considered suitable. The BPSolver team managed to complete at least one solution for each of the benchmarks. In addition to CLP(FD) and tabling, this achievement is also attributed to B-Prolog's foreach and list-comprehension constructs, which made concise encodings of many problems possible.

This article aims to present the main ideas of BPSolver's winning and hopeful solutions. The limit of space does not permit inclusion of the problem descriptions and the complete solutions. The interested reader is referred to the Web site `www.mat.unical.it/aspcomp2011/` for the problem descriptions, and `www.sci.brooklyn.cuny.edu/~zhou/asp11/` or `www.probp.com/asp11/` for the BPSolver's complete solutions. An overview of the language features of B-Prolog used in the solutions can be found in [7].

2 The Winning Solutions

Table 1 gives the benchmarks (ordered by score) that were won by BPSolver. The benchmarks marked with $*$ were won with a tie by more than one solver. Our solution to the *Hanoi Tower* problem was submitted a few hours after the due time due to a misunderstanding of the time zone.[1]

Three of the benchmarks were won by BPSolver with plain Prolog only. The *Grammar-Based-IE* program is a parser in Prolog for parsing Web documents. Although BPSolver's win of this benchmark was expected, the competition result

[1] The *Model & Solve* competition took more than two months than initially planned to complete and the organizers kindly forgave the tardiness of the submission.

showed that Clasp, which earned 80 points, was close as a parsing tool for large documents. The *Labyrinth* program solves the puzzle as a state-space search problem with hill-climbing and randomization. BPSolver actually solved 2 fewer instances than Clasp (12 vs 14) but it got a higher score because it was much faster than Clasp on the solved instances. The *Tomography* program solves the network covering problem by selecting a subset of nodes that has the largest coverage. This strategy is programmed into the BPSolver solution, and would be hard to express as a labeling strategy for CP and SAT solvers.

Five benchmarks were won by BPSolver with tabling. The *Reachability* program tests if a given node is reachable from a start node in a given directed graph. The BPSolver's solution is as follows:

```
:-table reach/1.
reach(X):-
    start(X).
reach(Y):-
    reach(X),
    edge(X,Y).
```

This problem is not trivial since the tested graphs were very large.

The *Hydraulic Planning* and *Hydraulic Leaking* are two path-finding problems. Given a graph, the *Planning* problem is to find a shortest sequential plan to pressurize a given set of jet nodes by connecting them to some full tank nodes. The *Leaking* problem is to find a shortest plan among those using the least number of leaking valves. BPSolver's solutions to these two problems are very short, thanks to mode-directed tabling in B-Prolog. The following shows our solution to the *Leaking* problem.

```
:-table pressurize(+,-,min).
pressurize(Node,Plan,(Leaks,Len)):-
    full(Node),!, Plan=[],Leaks=0,Len=0.
pressurize(Node,[Valve|Plan],(Leaks,Len)):-
    link(AnotherNode,Node,Valve),
    \+ stuck(Valve),
    pressurize(AnotherNode,Plan,(Leaks1,Len1)),
    Len is Len1+1,
    (leaking(Valve)->
        Leaks is Leaks1+1
     ;
        Leaks is Leaks1
     ).
```

For each plan, two attribute values are computed: `Leaks` is the number of leaking valves in the plan and `Len` is the length of the plan. Because only one argument can be optimized in B-Prolog, this program minimizes the compound argument `(Leaks,Len)`.

The *Airport Pickup* problem is another path-finding problem for which tabling seems more suitable than ASP. The problem is to find a plan for a set of vehicles

to move passengers between two airports in a city. Of course, a vehicle cannot move if the gas tank is empty. The most important planning task is to move a vehicle from one location to another. This task is described easily as a tabled predicate that finds a path that yields the maximal gas level in the vehicle.

BPSolver's solution to the 4-peg Hanoi Tower problem was an exciting one because it resulted from several failed attempts to use CLP(FD) and planning languages for the problem. Given two snapshots from the sequence generated by the Frame-Stewart algorithm [6], the problem is to find a sequence of moves to transform one state to the other. The solution, part of which shown below, is quite simple.

```
:-table plan4(+,+,+,-,min).
plan4(N,_CState,_GState,Plan,Len):-N=:=0,!,Plan=[],Len=0.
plan4(N,CState,GState,Plan,Len):-
   reduce_prob(N,CState,GState,CState1,GState1),!,
   N1 is N-1,
   plan4(N1,CState1,GState1,Plan,Len).
plan4(N,CState,GState,Plan,Len):-
   partition_disks(N,CState,GState,ItState,Mid,Peg),
   remove_larger_disks(CState,Mid,CState1),
   plan4(Mid,CState1,ItState,Plan1,Len1), % sub-prob1
   remove_smaller_or_equal_disks(CState,Mid,CState2),
   remove_smaller_or_equal_disks(GState,Mid,GState2),
   N1 is N-Mid,
   plan3(N1,CState2,GState2,Peg,Plan2,Len2), % sub-prob2
   remove_larger_disks(GState,Mid,GState1),
   plan4(Mid,ItState,GState1,Plan3,Len3), % sub-prob3
   append(Plan1,Plan2,Plan3,Plan),
   Len is Len1+Len2+Len3.
```

The subgoal `plan4(N,CState,GState,Plan,Len)` searches for a shortest plan to transform the current state `CState` to the goal state `GState`, where `N` is the number of disks in the problem. If `N`=0, the problem is solved. Otherwise, if the largest disk is in its goal position, the problem is reduced by removing the disk. If the largest disk is not in its final position, an intermediate state is generated such that the smallest `Mid` disks form a tower on `Peg` and the larger disks remain as in `CState`. The problem is then divided into three sub-problems: (1) build a tower of the smallest `Mid` disks on `Peg`; (2) move the disks larger than `Mid` to their final positions without using `Peg`; and (3) move the smallest `Mid` disks from the intermediate state to the goal state. Since the partition number `Mid` is known [5] and the peg on which the tower of the smallest `Mid` disks is to be built can be determined easily, the program involves no guessing and hence takes polynomial time.

Six benchmarks were won by BPSolver with CLP(FD). The *Tangram* is an old Chinese puzzle whose objective is to build a shape with seven pieces such that no two pieces overlap. Only 13 convex shapes can be built from the pieces.

As the search space is small, all the participated solvers got 100 in this benchmark. The *Weight-Assignment Tree* problem was inspired by query optimization in Database. Given a set of nodes, the objective of the problem is to build a tree that satisfies certain constraints. The `element` constraint is used in the solution for accessing elements of a collection with variable indices. The other four benchmarks (*Magic Square*, *Knight Tour*, *Disjunctive Scheduling*, and *Maximal Clique*) won by BPSolver all have well-known CSP encodings. The `foreach` and list-comprehension constructs of B-Prolog greatly facilitate the description of them. For example, the following defines semi and normal magic sets:

```
semi(Board,N,Magic):-
    foreach(I in 1..N, sum([Board[I,J] : J in 1..N])#=Magic),
    foreach(J in 1..N, sum([Board[I,J] : I in 1..N])#=Magic).

normal(Board,N,Magic):-
    semi(Board,N,Magic),
    sum([Board[I,I] : I in 1..N]) #= Magic,
    sum([Board[I,N-I+1] : I in 1..N]) #= Magic.
```

They would require five times as much code if only recursion were allowed.

3 Hopeful Solutions

Many of BPSolver's solutions have rooms for further improvement. Several solutions could have won had we put little further efforts into them. We show two such solutions.

The first one is the graph coloring problem. The BPSolver solution ranked last with only 15 points among the five participated solvers. The BPSolver solution uses the normal model in which a domain variable is used for each vertex in the given undirected graph and the domain is $0..K-1$, where K is the number of available colors. Instead of directly posting the constraint $C_i \neq C_j$ for each pair of adjacent vertices i and j, the BPSolver solution uses the built-in `post_neqs(`L`)` to post the disequality constraints. This built-in extracts complete subgraphs from the graph and posts an `all_distinct` constraint for each complete subgraph. Experiments have shown that `post_neqs` performs better than posting the disequality constraints separately.

The BPSolver solution, however, exploits no symmetry in the problem. It has been found that symmetry-breaking is very effective for graph coloring [3]. To implement the symmetry-breaking method, we extended `post_neqs` to let it return the list of extracted complete subgraphs.[2] With this built-in, the segment of the program for posting constraints and labeling variables can be rewritten into the following:

```
    ...  % create Vars and Neqs
    post_neqs(Neqs,Cliques),
```

[2] The built-in `post_neqs(`L`,`$Cliques$`)` is available in version 7.5#2.

```
largest_clique(Cliques,LClique),
(labeling(LClique)->labeling_ffc(Vars),!;fail),
...  % output
```

The subgoal `largest_clique(Cliques,LClique)` retrieves from `Cliques` a largest clique `LClique` where all the variables have the same domain, and the subgoal `labeling(LClique)` forcibly labels the variables in `LClique` with integers 0, 1, ..., and $m-1$, where m is the size of `LClique`. While the original BPSolver solution solved only 3 of the 15 instances under the time limit of 600 seconds, this improved version with symmetry-breaking solved all the 15 instances easily. This improved version would have won the benchmark since no solver solved all the instances in the competition.

The second hopeful solution is the *Sokoban Optimization* problem. The BPSolver solution treats the Sokoban problem as a generalized shortest path problem. For a state, if it is the goal state in which every box is in a storage location, it is done. Otherwise, the program chooses an intermediate state and splits the problem into two subproblems, one transforming the current state to the intermediate one and the other transforming the intermediate one to the goal state. All the states are tabled so that the same subproblem is solved only once. The following shows the main predicate of the program:

```
:-table plan_sokoban(+,+,-,min).
plan_sokoban(_SokobanLoc,BoxLocs,Plan,Len):-
    goal_reached(BoxLocs),!,
    Plan=[],Len=0.
plan_sokoban(SokobanLoc,BoxLocs,[push(BoxLoc,Dir,DestLoc)|Plan],Len):-
   select(BoxLoc,BoxLocs,BoxLocs1),
   neib(PrevNeibLoc,BoxLoc,Dir),
   \+ member(PrevNeibLoc,BoxLocs1),
   neib(BoxLoc,NextNeibLoc,Dir),
   good_dest(NextNeibLoc,BoxLocs1),
   reachable_by_sokoban(SokobanLoc,PrevNeibLoc,BoxLocs),
   choose_dest(BoxLoc,NextNeibLoc,Dir,DestLoc,NewSokobanLoc,BoxLocs1),
   insert_ordered(DestLoc,BoxLocs1,NewBoxLocs),
   plan_sokoban(NewSokobanLoc,NewBoxLocs,Plan,Len1),
   Len is Len1+1.
```

The subgoal `plan_sokoban(SokobanLoc,BoxLocs,Plan,Len)` finds a plan `Plan` with the minimal length `Len` for the current state, where `SokobanLoc` is the location of the warehouse keeper and `BoxLocs` is a list of box locations. The list `BoxLocs` is sorted in lexicographic order to make tabling more effective. When the goal has been reached (`goal_reached(BoxLocs)` succeeds when every box is in a storage location), an empty plan is returned. Otherwise, the second rule selects a box location `BoxLoc` from `BoxLocs` and a destination location `DestLoc` that can be reached from `BoxLoc` in the direction `Dir`, and adds the action `push(BoxLoc,Dir,DestLoc)` into the plan. A detailed description of the solution is given in [8].

The BPSolver solution solved 11 of 15 instances and failed to solve the remaining four instances due to lack of table space. The winner of this benchmark,

Clasp, solved all the instances. On the solved instances, however, BPSolver was actually faster than Clasp.

The BPSolver solution basically explores all possible states including deadlock states that can never occur in an optimal solution. Domain knowledge can be used to reduce the search space and filter out deadlock states [4]. We expect our solution to perform significantly better once domain knowledge is introduced.

4 Observations

The participation of BPSolver in the competition created a great opportunity to directly compare top-down tabled evaluation with bottom-up evaluation of logic programs, and CLP(FD) with SAT-based ASP solvers. BPSolver won almost all the path-finding and reachability-testing benchmarks. These results show that tabling as implemented in B-Prolog is competitive with the bottom-up grounders such as Gringo used in Clasp. Currently, the tabling system of B-Prolog natively supports computation of `min` and `max` aggregates. It can be enhanced with constructs for computing other aggregates such as `sum` and `count`. With these constructs, the *Company Control* and its related benchmarks can be described more naturally and solved more efficiently.

In comparison of CLP(FD) with SAT-based ASP solvers, the encodings in ASP are in general shorter than their counterparts in CLP(FD) thanks in part to the iterative fixpoint-computing procedure used in the grounder. Nevertheless, there are cases where the completion semantics of ASP requires lengthy descriptions. For example, for the *Airport Pickup* benchmark, one needs to describe such constraints as "an empty vehicle cannot drop a passenger" and "a non-empty vehicle cannot pickup another passenger". In CLP(FD), single assignment variables make the description of these kinds of constraints unnecessary. Loop constructs, such as `foreach` and list comprehension in B-Prolog, significantly ease modeling. As modeling languages of constraint satisfaction problems, CLP(FD) and ASP have differences but they are not important. For planning problems that do not have simple CSP encodings, however, ASP and dynamic programming tend to be more suitable than CLP(FD).

As solving tools, CLP(FD) and ASP have important differences. ASP benefits greatly from the SAT-solving techniques such as unit propagation, clause learning, conflict-directed backtracking, automated labeling heuristics, and randomized restart. An ASP programmer needs to give a declarative description of the problem and tune the solver on the problem to get the best setting. Tuning tends to require expertise, energy, and time. CLP(FD), on the other hand, relies on propagation and search to solve a problem. The system does not learn from failures during search and only loyally follows the specified strategy to select variables and values in labeling. For problems that have good global constraints (e.g., *Knight Tour* and *Disjunctive Scheduling*) and clear labeling strategies (e.g., *Maximal Clique* and *Tomography*), CLP(FD) does have its advantages over ASP.

BPSolver scored zero in six benchmarks, namely, *Partner Units P*, *Partner Units NP*, *Reverse Folding*, *Strategic Companies*, *Company Controls Optimize*,

and *Generalized Slitherlink*. The solution to *Generalized Slitherlink* was disqualified due to a bug in the `path_from_to` constraint, which was newly introduced into B-Prolog for the competition. A rerun of the corrected solution showed that the solution could have won 70 points. Our solution to *Partner Units* failed to distinguish between polynomial and NP cases, and the model used was not the best known in the literature (e.g., [1]). *Reverse Folding* can be treated as a path finding problem, and a dynamic programming encoding could perform better than our CSP encoding. *Strategic Companies* is intrinsically a satisfiability problem, and for such a problem CLP(FD) is hardly competitive with any decent SAT solvers. This result manifests the need to integrate CLP(FD) with a SAT solver. As mentioned above, the *Company Controls Optimize* problem and its decision problem can benefit from new tabled constructs for computing aggregates. BPSolver also performed poorly in *Stable Marriage*, *Solitaire*, and *Maze Generation*. Clasp scored 601 points in these 9 benchmarks, but BPSolver got only 82 points. Had BPSolver had the same performance as Clasp on these problems, it would have become No.1 in the overall ranking.

Acknowledgement

We would like to thank the organization committee at the University of Calabria for organizing the competition, and we especially are thankful to Giovambattista Ianni, Francesco Ricca, and Francesco Calimeri for patiently answering all our questions.

References

1. Markus Aschinger, Conrad Drescher, Gerhard Friedrich, Georg Gottlob, Peter Jeavons, Anna Ryabokon, and Evgenij Thorstensen. Optimization methods for the partner units problem. In *CPAIOR*, pages 4–19, 2011.
2. David Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–14, 1962.
3. Allen Van Gelder. Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230–243, 2008.
4. Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artif. Intell.*, 129(1-2):219–251, 2001.
5. Michael Rand. On the Frame-Stewart algorithm for the Tower of Hanoi. Technical report, Boston College (`https://www2.bc.edu/~grigsbyj/Rand_Final.pdf`).
6. B. M. Stewart and J. S. Frame. Problems and solutions: Advanced problems: Solutions: 3918. *American Mathematical Monthly*, 48:216–219, 1941.
7. Neng-Fa Zhou. The language features and architecture of B-Prolog. *TPLP*, Special Issue on Prolog Systems, 2011.
8. Neng-Fa Zhou and Agostino Dovier. A tabled Prolog program for solving Sokoban. In *submitted*, 2011.