On the Effectiveness of Looking Ahead in Search for Answer Sets

Guohua Liu and Jia-Huai You

Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada

Abstract. Most complete answer set solvers are based on DPLL. One of the constraint propagation methods is the so-called *lookahead*, which has been somewhat controversial, due to its high overhead. In this paper, we show characterizations of the problems for which lookahead is ineffective, and demonstrate, experimentally, that for problems that lie in the phase transition regions, search efficiency can be improved significantly by lookahead. This understanding leads to the proposal of a mechanism called *adaptive lookahead*, which decides when lookahead should be invoked dynamically upon learned information. This new mechanism is shown promising by the experiments.

1 Introduction

Most complete answer set solvers are based on DPLL search [2]. the constraint propagation mechanism *unit propagation*, also called *boolean constraint propagation* (BCP) [3], is considered the most important space prunning technique to improve the DPLL search. In answer set solver *smodels*, the algorithm for BCP is called the *Expand* function.

One deductive mechanism based on BCP/*Expand* is *lookahead*- before a decision on a choice point is made, for each atom, if fixing the atom's truth value leads to a contradiction, the atom gets the opposite truth value. In this way, such an atom gets a truth value from the truth value propagation of the already assigned atoms without going through a search process.

Lookahead, however, incurs high overhead[10] and has been shown ineffective in some SAT solvers[6]. The high pruning power, along with non-ignorable overhead, has made lookahead a controversial technique. There are two camps of constraint solvers. In one of them lookahead is employed during the search and in the other it is not.

This paper investigates the question on the effectiveness of lookahead, namely how to exploit its pruning power and avert the unnecessary overhead. We choose the well konwn answer set solver *smodels* as our experimental system. We report the following findings. First, we show some characterizations of the programs for which lookahead is ineffective, and identify representative benchmarks in which the situation arises. Second, we show experimentally that lookahead may benefit the search of the hard problem instances especially those in their phase transition regions. Third, we propose a mechanism called *adaptive lookahead*, which turns lookahead on and off dynamically upon

learned information. We implemented adaptive lookahead in smodels, which adapts well to different search environments it is going through.

Next section provides the background. Section 3 presents some characterizations by which we identify problems that run much slower with lookahead. Section 4 introduces the algorithm of adaptive lookahead. Section 5 provides experimental results. Finally, Section 6 provides a summary and comments on future directions.

2 Constraint Propagation in Smodels

The class of logic programs considered here is that of normal programs, which are collections of program rules of the form $h \leftarrow a_1, ..., a_m, \textbf{not}\ b_1, ..., \textbf{not}\ b_n$, where h, a_i, $1 \leq i \leq m$ and b_i, $1 \leq i \leq n$ are ground atoms, or *positive literals* and **not** b_i are *negative literals*. For an atom a, $not(\textbf{not}\ a)$ is identical to a. The *answer sets* of a program are defined in [5].

A set of literals is *consistent* if there is no atom a such that a and **not** a are both in the set, otherwise there is a *conflict*. A *partial assignment* is a consistent set of literals.

Let us define a *choice point* as a point during search where the branching heuristic picks a literal. This is also referred to as *making a decision*. Smodels performs constraint propagation before a decision is made. When lookahead is not involved, constraint propagation is carried out by a function called $expand(P, A)$, where A is a partial assignment. When lookahead is employed, constraint propagation is carried out as follows: for each unassigned atom a, assumes a truth value for it. If this leads to a conflict, then a gets the opposite truth value. This process continues, repeatedly, until no atom can be fixed a truth value by lookahead. Truth values are propagated in lookahead by $expand(P, A)$, which returns a superset of A, representing the process of propagating the values of the atoms in A to some additional atoms. For the details of $expand(P, A)$, the readers are referred to [9].

3 Situations where Overhead Dominates

Easy sub-program When searching for the solution of a program, it could be the case that no prunning is ever generated by lookahead in the course of solving some parts of it.This phonomenon could be caused by easy sub-programs discribed as follows.

Let P' and P be two ground programs, P' is called a *sub-program* of P if $P' \subseteq P$(with respect to the set of rules). A program P is called a *easy program* if any paritial assignment can be extended to a solution. In the process of solving a easy program the lookahead is totally wasted in that, if $expand(P, A) = A$ then $lookahead(P, A) = A$, for any partial assignemnt A generated during the search.

A hard program may contain some easy sub-programs. For example, a pigeon-hole program(the program modeling the pigion-hole problems) is known to be hard when the number of holes is smaller than the number of pigeons by 1. But it has an easy sub-program which can be got by removing some facts about piegons so that the resulting number of pigeons equals to the holes. The lookahead in this sub-program is totally useless.

Spurious pruning When lookahead finds a conflict, some search space is pruned. But some pruning may be immaterial to the rest of the search. Suppose, by an invocation of lookahead, a literal, say l, is added to the current partial assignment A. The addition of l may not contribute to further constraint propagations. This can be described by the following equation:

$$expand(P, A) \cup \{l\} = expand(P, A \cup \{l\}).$$

In this case, the lookahead is unnecessary since the decision on l can be delayed to any later choice point.

We can use the number of calls to the *Expand* function during search to measure the effectiveness of lookahead. Let us use N_{lh} and N_{nlh} to denote the number of calls to *Expand* in smodels with and without lookahead respectively. It is easy to see, for a given program P, if all the prunings by lookahead are spurious, then $N_{nlh} \leq N_{lh}$.

4 Adaptive Lookahead

The *adaptive lookahead* is designed to avoid lookahead when the search going through easy sub-programs or the spurious pruning happens frequently.

Two pieces of information can be used for this purpose. One is the number of failed literals(literals whose addition to the current partial assignment cause conflicts by expand). Note that a failed literal does not necessarily cause a backtrack since it could be the case that the negation of the failed literal is consistent with current partial assignment; Another is the number of dead-ends(where a backtrack is needed) detected during the search. The idea is that if after some runs, failed literals have been rarely found, it is likely that the search is in a space where pruning is insignificant, and will remain to be so for sometime to come, so lookahead is turned off; if dead-ends have been frequently encountered after some runs, it is likely that the search has entered into a space where pruning can be significant, so lookahead is turned on.

We call the resulting system *adaptive smodels* The control of lookahead is realized by manipulating two scores, $look\_score$ and $dead\_end\_counter$. The $look\_score$ is initialized to be some positive number, then deducted each time lookahead does not detect any conflict. When it becomes zero, lookahead will be turned off. The $dead\_end\_counter$ is initialized to be zero and increased each time a dead-end is encountered. Lookahead will be turned on if $dead\_end\_counter$ reaches some threshold. The counters will be reset after each turn. In this way, lookahead will be enabled only when some dead-ends are encountered by the search. So the spurious pruning is avoided.

In addition to the on/off control, lookahead will never be used in late search processes if it cannot detect any conflict after a number of atoms have been assigned. This is because the search efficiency cannot be improved much by lookahead if the conflicts it detects only happen in late stages of the search. The lateness is measured by a ratio of number of assigned literals to the number of literals in the program.

The initial value for $look\_score$, the amounts of increase and decrease, and the thresholds are determined empirically. We set the amount of increase/decrease to 1, $look\_score$ to 10, the thresholds of $dead\_end\_counter$ and $ratio$ to 1 and 0.8 respectively.

5 Experiments

The experiments serve three purposes. First, they confirm our findings of the problems where the performance is significantly deteriorated by the use of lookahead; second, they show that lookahead tends to be very effective for hard programs, especially those lie in the known regions of phase transition; and third, adaptive lookahead behaves as if it "knows" when to employ lookahead and when not to.

We run *smodels 2.32* with, without lookahead and with adaptive lookahead respectively. All of the experiments are run on Red Hat Linux AS release 4 with 2GHz CPU and 2GB RAM.

5.1 Cases where lookahead improves the search efficiency

Graph coloring We use Culberson's flat graph generator [1] to generate graph instances. In these graphs, each pair of vertices is assigned an edge with independent identity probability p. We use the suggested value of p to sample across the "phase transition". The number of colors is 3 and vertices of the graph is 400. The cutoff is 3 hours. For each measure point, we generate 100 instances and the average running time is reported.

The experiments show that lookahead drastically speeds up the search in the hard region. In the easier regions, the effectiveness of lookahead is insignificant(Fig. 1). The savings by lookahead can also be measured by the number of calls to *Expand*(Fig. 2.[1])

Random 3-SAT Another problem with well-kown phase transition is random 3-SAT. We test instances around its known phase transition region[8]. The results are very similar to graph coloring problems. The data are omitted for the sake of space.

Blocks-world The blocks-world problem is a typical planing problem. The instances we use are generated as follows. For n blocks, b_1, ..., b_n, the initial configuration is b_1 on the table and b_{i+1} on b_i for $1 \leq i \leq n-1$. The goal is b_n on the table, b_1 on b_n and b_{i+1} on b_i for $1 \leq i \leq n-2$. Under this setting, each block in the initial state has to be moved to get to the goal, so the problem turns out to be nontrivial. The minimum steps needed is $2n-2$.

Grippers The Grippers problem is another intensively studied planning problem. The goal of this problem is to transport all the balls from room $R1$ to room $R2$. To accomplish this, a robot is allowed to move from one room to the other, pick up, and put down a ball. Each gripper of the robot can hold one ball at a time.

In our experiments two kinds of settings are used. In the first, all of the balls are in $R1$ initially and $R2$ in the goal state. In the second, there are equal number of balls in $R1$ and $R2$ initially and in the goal state, the balls initially in $R1$ are in $R2$ and initially in $R2$ are in $R1$.

Lookahead appears to be very helpful to both of these planning problems, especially when the instances are hard(Table 1, 2).

[1] In comparison, the number of choice points is usually not a good indicator, as a reduction may be achieved in the expense of a huge overhead.

5.2 Cases where lookahead reduces search efficiency

We choose the pigeon-hole problem where the number of pigeons is greater than holes by 1 and Hamiltonian cycle problem over complete graphs as the representatives of programs with easy sub-programs and spurious pruning respectively. The experiments show lookahead hugely degrades the search(Table 3, 4).

5.3 Adaptive smodels

The adaptiveness of adaptive smodels is clearly shown by the experiments: for problems where lookahead helps, adaptive smodels performs as well as or even better than smodels(Figure 1, 2 and Table 1, 2). For the problems where the overhead of lookahead dominates, adaptive smodels works largely as smodels without lookahead(Table 3, 4).

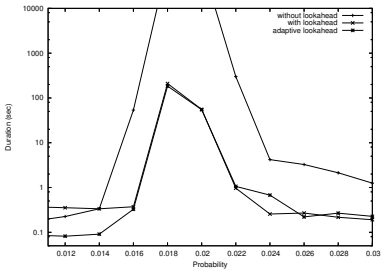

Fig. 1. Runing time

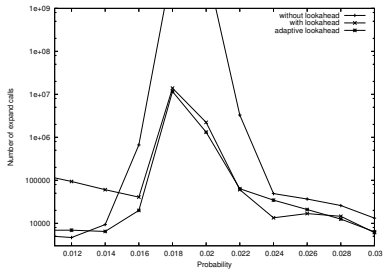

Fig. 2. Number of $expand$ calls

b	s	No Lookahead	Lookahead	A-Lookahead
14	26	165.88	30.49	7.34
	25	359.65	5.45	5.46
15	28	335.11	53.07	10.54
	27	4673.35	7.35	7.35
16	30	375.20	6276.28	14.66
	29	8585.08	10.15	10.50
17	32	1197.65	145.59	21.24
	31	701.86	16.48	16.40
18	34	–	145.48	29.28
	33	–	21.42	21.39

Table 1. Blocks-world Problem

R1	R2	s	No Lookahead	Lookahead	A-Lookahead
4	0	7	0.1	0.43	0.41
		6	0.09	0.17	0.17
5	0	11	8.31	77.35	64.94
		10	1824.55	189.40	97.12
6	0	11	263.75	1117.02	1084.31
		10	2345.49	490.93	487.64
3	3	11	0.70	15.80	15.98
		10	77.93	15.64	15.52
4	4	12	1749.35	419.42	412.64
		11	5463.57	175.61	175.73

Table 2. Gripper problem

6 Summary and Future Directions

In this paper, we show that lookahead could be a burden as well as an accelerator to the DPLL search. We analyze why lookahead sometimes slows down the search and characterize the reasons as embeded easy sub-programs and spurious pruning. Based on the analysis, we propose an adaptive lookahead mechanism, which takes the advantage of lookahead while avoiding the unnecessary overhead caused by it.

p	h	No Lookahead	Lookahead	A-Lookahead
5	4	0.00	0.01	0.01
6	5	0.00	0.02	0.01
7	6	0.02	0.06	0.02
8	7	0.13	0.49	0.11
9	8	1.18	4.38	0.94
10	9	12.10	43.66	9.78
11	10	137.06	480.19	112.47
12	11	1608.41	5439.09	1343.93

Table 3. Pigeon hole

n	No Lookahead	Lookahead	A-Lookahead
50	2.25	64.92	8.57
60	3.91	183.59	19.74
70	6.31	467.28	40.18
80	9.58	986.85	74.02
90	13.77	1862.86	123.66
100	19.12	3522.24	194.24
110	25.56	6129.59	288.53
120	33.36	7255.97	412.58

Table 4. Hamiltonian cycle

The (in)effectiveness of lookahead in SAT solvers were studied in [6]. The main conclusion is lookahead does not pay off when integrated with look-back techniques. Our research here focus on the effect of lookahead on a SAT-like solver without any look-back. Basically, we only compare the solver with lookahead to the pure boolean constraint propagation(unit propagation). We show that for such a solver, lookahead can significantly improve the search for the extremely hard instances.

Some recent answer set solvers [4, 7] adopt look-back techniques. The effect of lookahead on them and the better way to integrate lookahead with them, e.g., *selective lookahead*: instead of the entire set of unassigend atoms, some subset of it may be selected by lookahead for testing, are topics interested us.

References

1. J. Culberson. http://web.cs.ualberta.ca/joe/coloring/index.html.
2. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.
3. Jon Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, 1995.
4. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proc. IJCAI'07*, pages 386–392, 2007.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. 5th ICLP*, pages 1070–1080, 1988.
6. Enrico Ginuchiglia, Marco maratea, and Armando Tacchella. (in)effectiveness of look-ahead techniques in a modern sat solver. In *Proc. CP'03*, pages 842–846, 2003.
7. J.Ward and JSchlipf. Answer set programming with clause learning. In *Proc. ICLP'04*, pages 302–313, 2004.
8. D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. In *Proc. of AAAI'92*, pages 459–465, 1992.
9. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2), 2002.
10. J. You and G. Hou. Arc consistency + unit propagation = lookahead. In *Proc. ICLP'04*, pages 314–328, 2004.