DSatz: A Directional SAT Solver for Planning

Mark Iwen and Amol Dattatraya Mali
Electrical Engineering and Computer Science
University of Wisconsin, Milwaukee, WI 53211
iwen2724@uwm.edu, mali@miller.cs.uwm.edu, Fax: 1-414-229-2769

Abstract

(Appears as a regular paper in the proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI), IEEE Computer Society, Washington D.C, Nov. 2002, pp. 199-208.)

SAT-based planners have been characterized as disjunctive planners that maintain a compact representation of search space of action sequences. Several ideas from refinement planners (conjunctive planners) have been used to improve performance of SAT-based planners or get a better understanding of planning as SAT. One important lesson from refinement planning is that backward search being goal directed can be more efficient than forward search. Another lesson is that bidirectional search is generally not efficient. This is because the forward and backward searches can miss each other. Though effect of direction of plan refinement (forward, backward, bidirectional etc.) on efficiency of plan synthesis has been deeply investigated in refinement planning, the effect of directional solving of SAT encodings is not investigated in depth. We solved several propositional encodings of benchmark planning problems with a modified form (DSatz) of the systematic SAT solver Satz. DSatz offers 21 options for solving a SAT encoding of a planning problem, where the options are about assigning truth values to action and/or fluent variables in forward or backward or both directions, in an intermittent or non-intermittent style. Our investigation shows that backward search on plan encodings (assigning values to fluent variables first, starting with goal) is very inferior. We also show bidirectional solving options and forward solving options turn out to be far more efficient than other solving options. Our empirical results show that the efficient systematic solver Satz which exploits variable dependencies can be significantly enhanced with use of our variable ordering heuristics which are also computationally very cheap to apply. Our main results are that directionality does matter in solving SAT encodings of planning problems and that certain directional solving options are superior to others.

1 Introduction

Planning as satisfiability paradigm [5] yielded significantly improved results in plan synthesis. In this paradigm, a SAT instance is generated assuming that a planning solution of k steps exists such that this SAT instance has a solution if and only if there is a plan of k steps. The SAT instance compactly represents all action sequences of length k. A SAT planner can be viewed as a disjunctive planner. The impact of planning as SAT is clear from several papers on this paradigm.

Lot of developments in refinement planning (or conjunctive planning) have been shown to be relevant/useful in the SAT planning framework. Most conjunctive classical planners are unidirectional (forward state-space only or backward state-space only) including the recent and more efficient ones like HSP [2], HSP-r [3], VVPLAN [14] and UNPOP [9]. Bidirectional planners PRODIGY2 [10], NOLIMIT [12] and FLECS [13] are incomplete. Forward search and backward search missing each other or not meeting each other soon, is the most general way to describe the causes of incompleteness or inefficiency of these planners respectively. Some of the efficient conjunctive classical planners are HSP-r [3] and AltAlt [11]. Both these planners carry out backward state-space search.

Though SAT encodings are declarative representations and can be processed in a non-directional fashion which in fact can improve efficiency of plan synthesis, there are reasons to expect that (i) backward search on the encodings, being goal-directed would be more efficient than forward search and (ii) bidirectional search and forward search would be inferior, respectively because of the same reasons that make (i),(ii) happen in refinement planning. Different truth assignments of action variables in a SAT encoding can be viewed as different partial plans in refinement planning.

Let us consider the first lesson of superiority of backward search from refinement planning. Since arities of actions are typically higher than arities of fluents, the number of fluent variables is less than the number of action variables. For example, maximum arity of action in transportation logistics domain is 3 (in load(x,y,z) and unload(x,y,h) and fly(y,z,h)) and maximum arity of fluent is 2 (in in(x,y), at(x,z) etc.). With p packages, c cities and n planes, there will be $O(k.(p.c.n+n.c^2))$ action variables and $O(k.(n.c+n.p+p.c))$ fluent variables in a k step state-space encoding. This greater than relationship between maximum action arity and maximum fluent arity also applies to blocks world when operator $move(x,y,z)$ and predicate $on(x,y)$ are used. $move(x,y,z)$ moves block x from top of block y or table to top of block z or table, $x \neq y, y \neq z, x \neq z$. It is thus worth investigating whether assigning values to fluent variables first, starting from goal yields an increase in planning efficiency, because this is like being goal directed and having lower branching factor in the search space of a SAT solver.

Consider the second lesson of inferiority of bidirectional search. Assigning values to action variables and fluent variables in solving SAT encodings in bidirectional fashion can be inferior in general or at least not work as good as unidirectional search because the forward and backward searches can miss each other. Whether (i), (ii) hold in SAT planning is not clear. This is because the disjunctive representation, when solved in backward direction, may not undergo significant simplification, increasing the solving times. The compact nature of the disjunctive representation may make it easier for forward search and backward search to meet fast or detect failures early, making bidirectional search more efficient. Whether (i), (ii) hold in SAT planning is thus worthy of investigation since this can make development of SAT solvers specialized for solving planning problems possible. To achieve this, we modified the efficient systematic SAT solver *Satz* [8] to implement DSatz (Directional Satz) which assigns values to action and/or fluent variables in forward and/or backward directions with an option of intermittent assignment (alternating between forward and backward passes). We evaluate DSatz with several options on various benchmark problems to investigate lessons (i) and (ii). We show that forward search where both action and fluent variables are assigned values in forward direction performs orders of magnitude faster than backward search where both action and fluent variables are assigned values in backward direction. We also show that bidirectional search where both action and fluent variables are assigned values in both directions gives better results than backward search. Many directional solving options of DSatz perform better than Satz.

Our work makes the following contributions.

• A deeper investigation of variable ordering heuristics in SAT-based planning with the DSatz systematic solver that offers 21 directional solving options. These options can be viewed as different variable ordering heuristics that are computationally very cheap.

• We show that systematic solver Satz that efficiently exploits variable dependencies, does not necessarily solve plan encodings faster, despite the dependencies among variables in the encodings of planning and that it can be significantly improved with directional solving. This shows that directionality does matter in solving SAT encodings of planning.

• We show that performance of the options BABF and FARF is superior. We also show that increasing the number of subgoals in goal leads to more unit clauses in encodings which increase the encoding simplification opportunities and considerably enhance the performance of some backward solving options.

2 Background

In this section, we first explain the working of SAT planners. Then we explain the state-space encoding [6], followed by explanation of the systematic solver Satz. We use STRIPS representation. An action is assumed to be a fully ground instance of an operator.

An encoding is composed of several types of constraints such that these are all satisfied if and only if there is a plan of k steps. Because of this, the encoding can be passed to any SAT solver like walk-sat which does local search [5] or Satz which does systematic search. If k chosen is less than the minimum plan length, k is increased and a new encoding is generated and solved. This procedure is repeated until solution is found. Plans of length lower than k can be found by making action variables at the extra time steps false, so that no actions making the plans invalid occur. We explain (and also empirically evaluate) the state-space encoding with explanatory frame axioms (explained in this section next) because of its superior performance [4], though other variants of state-space encodings exist.

The state-space encoding is based on the idea of using state of world to decide which actions can and cannot occur, from state-space refinement planning. The clauses in a state-space encoding capture the following constraints: (i) Any of the actions from the domain, if it occurs at any of the time steps t from the interval $[0, k-1]$, implies the truth of its preconditions at t and the truth of its effects at $(t+1)$. (ii) The initial state is true at time 0. (iii) The goal must be true at time k. (iv) Conflicting actions (one action deleting a precondition of another) cannot occur at the same time step. (v) Persistence of fluents is described by "explanatory frame axioms" which state that if the truth of a fluent f changes over an interval $[t, t+1]$, some action changing that truth must occur at t.

Satz is a variant of Davis-Putnam-Logemann-Loveland procedure (DPLL). Given a formula F, DPLL procedure recursively calls itself on $(F \wedge x)$ and if that does not yield the result "satisfiable", it calls itself on $(F \wedge \neg x)$, where x is a branching variable. DPLL procedure simplifies SAT instances by unit propagation. For example, $((a \vee \neg b) \wedge (c \vee b))$ can be simplified to a, if b is assigned true. Satz includes novel combinations of unit propagation and the MOM heuristic (maximum occurrences in clauses of minimum size). The brief explanation of how Satz works (that we provide next) is from [1]. Satz employs the following extensive one step look ahead. It has a series of filters that it employs. Satz considers T variables, where T is an empirically determined parameter, in the following way: It assigns true to a variable and finds no. of binary clauses yielded by unit propagation after true is assigned to the variable. Let these be p. Then it assigns false to the variable and it finds the number of binary clauses derived by unit propagation after false is assigned. Let these be n. After p, n values are found for all of T variables, it branches on variable with highest value of (p . n. 1024 + p + n) first. This allows satz to produce more balanced search trees. This score function penalizes variables that produced good subformulas under one assignment (true/false) but not the other (true/false).

3 Directional Solving

Assigning values to action variables before fluent variables can considerably simplify an encoding, as shown in [4]. This is because the truth of fluent variables is dependent on the truth of action variables. This reduces the search space of a SAT solver from $2^{O(|A|+|U|)}$ to $2^{O(|A|)}$, $\mid A \mid, \mid U \mid$ being the number of actions and fluents in the encoding respectively. Effects of directionality in solving encodings are not investigated in [4]. Though Satz has heuristics to do simplification of a SAT instance and these may in fact assign values by taking into account dependency between action and fluent variables, Satz does not necessarily solve encodings with the kinds of directional solving options we implemented. In fact our empirical results from Fig. 1 show that general SAT solvers like Satz can be improved by certain kinds of directional solving (e.g. bidirectional option BABF) for problems like classical planning. It is worth investigating a large number (21) of directional solving options, since different options can allow different amount of simplification of encodings. In the rest of this section, we first describe what the solving options in DSatz are and then explain how Satz was modified to follow them.

18 out of the 21 solving options can be captured in the form d_1xd_2y. Direction options d_1, d_2 may have one of the three values **F, R, B** that indicate forward, rear (backward) and bidirectional respectively. x, y may have the values A, F that indicate action variables and fluent variables respectively, with the constraint that x, y do not have same values, the value b being an exception. With the d_1xd_2y form of solving option, variables of type x are assigned values in direction d_1 and it is only after this assignment is done, that variables of type y are assigned values in d_2 direction.

In a k step encoding, initial state corresponds to time step index 0 and goal state corresponds to time step index k. FAFF option means that both action and fluent variables are to be assigned values in forward direction, such that action variables are assigned value first, in the increasing order of time step indices. After some action variable at each time step is assigned a value, fluent variables are assigned values, in increasing order of time steps, so that at each time step, there are some fluent variables that are assigned values. Passive variables are those which are assigned or those whose values are fixed by an assignment to other variables. For example, if $(\neg a \vee b)$ is a clause in the SAT encoding and a is assigned true, then b has to be assigned true. Then a and b both become passive variables. If a is set to false, it continues to be passive, but then b is no longer passive. With FARF option, action variables are assigned values in forward direction. After some action variables at each time step are assigned a value, some fluent variables at each time step are assigned values, in the decreasing order of step indices (thus following reverse direction). In particular, the first variable assigned is an action variable from time step 0. The second variable assigned is an action variable from time steps 0 or 1 unless more than 50 $\%$ of the total number of action variables from these two time steps are passive. If the 50 $\%$ limit is exceeded, a variable from time step 2 that is not passive is assigned. If all variables from time step 2 are passive, a variable from time step 3 that is not passive is assigned. The third variable assigned is an action variable from time steps 0 or 1 or 2 unless more than 33.33 $\%$ of the total number of action variables from these three time steps are passive. Once action variables are assigned values in this manner, fluent variables are assigned values in a similar manner. The solving options RAFF, RARF, FFFA, FFRA, RFFA and RFRA can be understood in a similar fashion.

Consider the option FABF. FA in this option has meaning similar to that in FAFF and FARF options explained earlier. BF means that fluent variables are assigned values in both directions such that some fluents at time 0 are assigned values, followed by some fluents at time k, followed by some fluents at time 1, followed by some fluents at time (k - 1) and so on. Thus starting with forward direction, the assignment of values to fluents in both directions is interleaved. In the option BABF, fluent variables are assigned values in a similar fashion to that in FABF and the action variables are assigned values so that some action variables at time 0 are assigned values, followed by assignment of values to some action variables at time (k - 1), followed by assignment of values to action variables at time 1, followed by assignment of action variables at time (k - 2) and so on. As per the semantics of d_1xd_2y, when BABF option is chosen, fluent variables are assigned values only after action variables are assigned values, that is, the assignment in BA is done. The options RABF, BAFF, BARF, BFRA, FFBA, RFBA, BFFA and BFBA can be understood in a similar fashion.

Second letter B in FB, RB and BB options stands for both (actions and fluents). First letter B in BB stands for both directions, that is, forward and backward. Note that FB is not same as FAFF or FFFA, RB is not same as RFRA or RARF and BB is not same as BABF or BFBA. With the FB solving option, some fluent variables at time 0 are assigned values, followed by assignment to some action variables at time 0, followed by assignment of fluent variables at time 1 and so on. The RB option is similar to FB with the difference that the direction of assignment is backward. The BB option is similar to FB, RB options with the difference that the assignment takes place in both directions (thus being intermittent). In terms of directionality in searching for solution, FB, RB and BB are respectively similar to forward state-space, backward state-space and bidirectional state-space refinement planning.

Satz was modified to choose branching variable only from the set of variables permitted by the solving option chosen. For example, if solving option is FAFF, DSatz branches only on one of the action variables at a time step in forward pass because of FA part of FAFF. Thus with FAFF solving option, after assigning an action variable at time t, DSatz branches only on action variable at $(t+1)$ if more than a certain number of action variables from steps 0, 1, 2, ... t are passive and if not all action variables at step t+1 are passive. Satz was not changed in any other way. Thus DSatz uses heuristics of Satz, to choose branching variable from multiple variables that meet criterion in the solving option.

4 Empirical Evaluation

Our empirical evaluation on various problems in benchmark domains is shown in Fig. 1,2,3,4 and 5. In Fig. 1,2,3 and 4, * denotes that the option had the lowest solving time on the problem, - denotes that the problem could not be solved in 1 hour and ** denotes that the option had the worst solving time on the problem. We did not simplify the encodings by unit propagation before feeding them to DSatz since DSatz simplifies the encodings like Satz. The encodings were generated by our code written in C. The encoding generation times were negligible in comparison with the encoding solving times. The encodings contained the following clauses: (i) unit clauses representing initial state and goal, (ii) clauses representing the truths of preconditions and effects of actions whenever they occur, (iii) clauses representing mutual exclusion relations between actions o_i and o_j if o_i deletes precondition of o_j, and, (iv) explanatory frame axioms. Blocks world, Transportation logistics, Rocket and Train are parallel domains. Monkey and Robot are serial domains. Multiple blocks whose tops are clear can be moved at same time in blocks world. There is only one operator in this domain ($move(x,y,z)$). Grippers are not represented in the problems and preconditions and effects of $move(x,y,z)$. This operator moves block x from top of block y or table to top of block z or table. Clearly, $x \neq y, y \neq z, x \neq z, x \neq Table$.

The first set of empirical results is shown in Fig. 1. First 18 solving options (from FAFF to BFBA) are in the form d_1xd_2y. The experiments were ran on an SGI O2 running Irix 6.5 (RM5200 processors (300 Mhz MIPS) with 128 MB RAM). The reported times are cpu seconds. The problem eight_b_t_i is problem of inverting a tower of 8 blocks. bw_large.a, rocket.a and rocket.b are benchmark problems from BLACKBOX distribution [7]. m1_logistics.a, m2_logistics.a are modified versions of logistics.a and m_rocket_ext.a is a modified version of rocket_ext.a. The 3 problems were modified (by removing objects) only to reduce encoding sizes, only because of insufficient memory. m1_logistics.a and m2_logistics.a are obtained from logistics.a by removing 2 and 3 packages from it respectively. m_rocket_ext.a has 5 cargo items. The encodings were automatically generated, by taking into account all actions in the domains. The number of steps in the encodings was set to the lowest value needed to find solution.

The evaluation in Fig. 1 shows that the BABF solving option gives better results than other solving options. Action and fluent variables are related and in fact one of these two kinds of variables can be eliminated from the encoding [6]. It is possible to solve the encodings containing both action and fluent variables by assigning values to only action variables or only fluent variables. Let us revisit the transportation logistics example in the introduction. Assigning values to fluent variables before action variables can reduce the size of search space from $2^{O(k.(p.c.n+n.c^2)+L)}$ to 2^L, where L is the number of fluent variables. This gives a reason to expect that assigning values to fluent variables first might work faster than other directional solving options. Backward search has been shown to be generally more efficient in refinement planning. However the option RB that closely resembles operation of backward state-space refinement search performs the worst.

We conducted a statistical analysis of the empirical results from Fig. 1. We discuss this below, followed by a discussion of simplification of SAT encodings. After this, we discuss empirical results from Fig. 2 and 3.

Satz chooses a variable for branching if it does not detect contradiction by using it. This holds for DSatz as well, since DSatz is a version of Satz where restrictions are put only to ensure directionality. It was found that DSatz chose fewest number of branching variables with BABF option and highest number of branching variables with the RB option. The performance of RB is inferior not only because of larger number of contradictions, but also because such contradictions are not detected early. On the problems that were solved by all options, BABF lead to lowest average solving time and RB lead to highest average solving time. On these problems, the average number of times DSatz chose branching variable with RB was 190 times higher than that for BABF. On these problems, the average solving time with BABF was 93 times lower than that for RB. On these problems, the average number of nodes in search tree with RB was 190 times higher than that for BABF. FB and BB lead to higher number of branching variables as well as larger number of nodes in tree than BABF.

11 out of 21 solving options (Fig. 1) lead to selection of fewer branching variables than Satz. These options also lead to trees containing fewer nodes than with Satz. 9 options (out of the 11 mentioned above) lead to lower average solving time than Satz, on problems from Fig. 1 that were solved by all options.

We discussed statistical analysis of the results above. To put the results in perspective, we next discuss the simplification that one can obtain by assigning values to action and fluent variables. Assigning values to action variables in forward direction can give lot of simplification. This is because the occurrence of an action also implies truth of its preconditions and effects which in turn generates several unit clauses that can simplify an encoding further. If true is assigned to an action variable $o_1(t)$, unique value can be assigned to a fluent variable that is its precondition (time t) or effect (time $(t+1)$). For example, if effects of o_1 are $g, \neg h$ and its preconditions are h, q, then $o_1(t) = true$ allows unit propagation yield $h(t) = true, q(t) = true, g(t+1) = true$ and $h(t+1) = false$. On the other hand, assigning values to fluents in backward direction only means that one can infer that either the truth of the fluent was preserved or that some action changing it occurred. In the example of o_1, the assignment $g(t+1) = true$ does not allow assignment of unique value to $o_1(t)$. This also holds for the assignments $h(t) = true, q(t) = true$ and $h(t+1) = false$. If unit clauses containing action variables are not derived, more simplification does not take place. This is correlated with the worst performance (solving option RB). Options containing d_1x = BA always give better performance because of the role of action variables in simplifying encodings and because BA means assigning values to such variables in both directions. BABF gives best performance (Fig. 1) both because of larger amount of simplification and faster failure detection that this directionality makes possible.

Results on additional problems are shown in Figures 2 and 3. This evaluation was conducted on a Dual Intel Pentium II 400 MHz SunOS 5.7 machine. First 18 solving options (from FAFF to BFBA) are in the form d_1xd_2y. Times reported are CPU seconds. Four options (RARF, RABF, RFRA and RFBA) that were found to be less efficient in the first set of experiments (Fig. 1), were not used in the second set of experiments reported in Fig. 2 and 3. The number of steps in the encodings were set to the lowest values needed to find solution (except robot.a in Fig. 3 explained later). Some problems in Fig. 2 and 3 are variants of benchmark problems. The encoding of robot.a (Fig. 3) had fewer steps than the minimum necessary to find solution, because we wanted to see whether directional solving worked well also on unsatisfiable instances.

For each solving option in Fig. 2 and 3, we found the following values (over 13 problems in Fig. 2 and 3) - (i) Average solution time, (ii) number of problems on which its solving time was worst, (iii) number of problems on which its solving time was best and (iv) the number of problems solved. We also found the (v) number of nodes in search tree and (vi) the number of contradictions detected, for each option on each problem. It was found that solving times were lower for options that made detection of larger number of contradictions possible. These options also lead to search trees with fewer nodes. Results on problems from Fig. 2 and 3 show that the option FARF had the lowest average solution time and had lowest solution time on more problems than all other options. Average solution times for FAFF and FABF were very close to that for FARF. These three options are the only ones that solved all 13 problems from Fig. 2 and 3. These three options also performed better than Satz. For these 3 options, the sizes of the search trees were lower and the number of contradictions detected were higher. All the six criteria (i, ii, iii, iv, v and vi) applied to options RAFF, RFFA and RB showed that these options were very inferior. This second set of experiments shows that assigning values to action variables in forward direction is a very good idea.

Initial states in classical planning problems are completely specified. Goal states are partially specified. As a result, initial state provides more unit clauses than goal. It is well known that an encoding can be simplified to greater extent if there are more unit clauses in it. Does this mean that the performance of backward solving options can improve when goal state is completely described or at least made stronger by adding subgoals? To evaluate this, we carried out an additional empirical evaluation. The results of this are shown in Fig. 4 and 5. This evaluation was conducted on a Sun Ultra 10 machine with 128 MB RAM. The problems used in this evaluation were obtained from the problems from Fig. 1,2 and 3 by manually adding more subgoals to goal. This increased the number of clauses. This did not increase the number of variables. We did not add any negated subgoals to goal. In blocks world problems, we specified positions of all blocks in goal to have more subgoals. In transportation logistics and rocket domains, we included locations of planes and trucks in goal. This

Problem	# Variables	# Clauses	**FAFF**	**FARF**	**FABF**
eight_b_t_i	4232	372625	69.19	69.19	69.33
bw_large.a	4518	520872	48.7 (*)	48.81	48.73
m1_logistics.a	4926	79024	168.29	127.01	127.29
m2_logistics.a	5586	89574	19181.68	20000.36	20322.91
rocket.a	1878	14068	1.7	1.75	1.76
rocket.b	1878	14068	1.82	1.96	1.89
m_rocket_ext.a	1078	8573	0.38	0.37 (*)	0.38

Problem	**RAFF**	**RARF**	**RABF**	**BAFF**	**BARF**
eight_b_t_i	69.46	69.49	69.51	69.57 (**)	69.53
bw_large.a	1292.7	1473.88	1474.68	48.86	49.24
m1_logistics.a	2243.14	2518.12	2522.74	267.93	168.29
m2_logistics.a	$>$ 12 hrs	$>$ 12 hrs	$>$ 12 hrs	2240.68 (*)	2728.45
rocket.a	21.24 (**)	20.6	20.59	1.81	1.79
rocket.b	19.36 (**)	19	18.37	1.89	1.96
m_rocket_ext.a	2.61 (**)	2.26	2.24	0.4	0.4

Problem	**BABF**	**FFFA**	**FFRA**	**FFBA**	**RFFA**
eight_b_t_i	69.48	69.26	69.35	69.21	69.23
bw_large.a	49.41	76.05	75.72	76.18	17102.07
m1_logistics.a	110.7 (*)	1400.45	624.06	704.86	3936.97 (**)
m2_logistics.a	2311.64	$>$ 12 hrs	23144.68	25753.39	$>$ 12 hrs
rocket.a	1.8	1.73	1.71	1.74	0.69 (*)
rocket.b	1.96	1.9	1.95	1.98	3.58
m_rocket_ext.a	0.4	0.38	0.38	0.38	0.5

Problem	**RFRA**	**RFBA**	**BFFA**	**BFRA**	**BFBA**
eight_b_t_i	69.29	69.31	69.35	69.32	69.31
bw_large.a	17166.99	17345.17	77.08	77.18	77.41
m1_logistics.a	2233.26	2445.68	1462.49	665.21	1631.07
m2_logistics.a	$>$ 12 hrs	$>$ 12 hrs	$>$ 12 hrs	23665.91	$>$ 12 hrs
rocket.a	0.7	0.7	2.05	2.1	2.08
rocket.b	3.48	3.67	2.25	2.37	2.24
m_rocket_ext.a	0.49	0.51	0.48	0.48	0.48

Problem	**FB**	**RB**	**BB**	**Satz**
eight_b_t_i	69.34	69.36	69.35	69.03 (*)
bw_large.a	77.62	17399.19 (**)	80.2	568.12
m1_logistics.a	712.44	3850.21	504.76	197.16
m2_logistics.a	37262.7	$>$ 12 hrs	36719.21	12216.57
rocket.a	1.79	0.93	1.85	1.25
rocket.b	1.87	4.62	1.95	1.24 (*)
m_rocket_ext.a	0.4	0.43	0.41	0.41

Figure 1. Empirical evaluation of directional solving of SAT encodings.

Problem	# **Variables**	# **Clauses**	**FAFF**	**FARF**	**FABF**
train.a1	1288	14513	0.27	0.27	0.27
train.a2	3875	73726	1.6	1.6	1.55 (*)
train.a3	3504	71908	646.2	662.39	672.19
bw_large.b1	8190	1209317	451.61	424.67 (*)	426.83
log-new.a	2612	20877	0.32	0.32	0.32
log-new.b	2318	17321	0.26 (*)	0.27	0.28
monkey.a	356	2949	0.04	0.04	0.04
monkey.b	945	9866	0.16 (*)	0.16 (*)	0.16 (*)
monkey.c	1908	34508	34.81	31.9 (*)	31.75

Problem	**RAFF**	**BAFF**	**BARF**	**BABF**	**FFFA**
train.a1	0.26	0.28	0.26	0.27	0.36
train.a2	-	1.75	1.63	1.64	1.98
train.a3	-	691.31	686.19	656.23	-
bw_large.b1	-	-	-	-	-
log-new.a	0.3	0.3	0.28 (*)	0.3	0.34
log-new.b	1.16	0.31	0.31	0.31	0.28
monkey.a	0.04	0.05	0.03	0.04	0.04
monkey.b	51.1	0.17	0.34	0.34	3.5
monkey.c	-	2133.73	1941.84	2158.64	1772.49

Problem	**FFRA**	**FFBA**	**RFFA**	**BFFA**	**BFRA**
train.a1	0.37	0.38 (**)	0.37	0.27	0.26
train.a2	1.97	1.97	-	1.7	1.6
train.a3	-	-	-	-	-
bw_large.b1	-	-	-	-	-
log-new.a	0.34	0.31	- (**)	0.34	0.3
log-new.b	0.27	0.27	0.29	1.21	1.22
monkey.a	0.04	0.04	0.06 (**)	0.04	0.04
monkey.b	3.48	3.58	150.37 (**)	2.99	2.96
monkey.c	1831.79	1907.47	-	1707.51	1745.39

Problem	**BFBA**	**FB**	**RB**	**BB**	**Satz**
train.a1	0.28	0.25 (*)	0.38 (**)	0.27	0.3
train.a2	1.56	1.98	-	1.92	1.74
train.a3	-	624.67 (*)	-	635.12	5569.4
bw_large.b1	-	-	-	-	$>$ 4 hrs
log-new.a	0.32	0.28 (*)	0.32	0.31	0.3
log-new.b	1.22 (**)	0.27	0.32	0.32	0.28
monkey.a	0.03	0.03	0.04	0.03	0.02 (*)
monkey.b	3.05	3.34	120.43	3.71	2.67
monkey.c	1740.76	1653.79	-	2169.64	1186.5

Figure 2. Evaluation of directional solving of SAT encodings. - denotes that problem could not be solved in 1 hour.

allowed us to derive multiple problems from a problem by controlling the number of subgoals added. These results show that the addition of subgoals drastically changed the performance of some directional solving options. To illustrate this, we have reported the ratios of solving times of some options with and without extra subgoals in Fig. 5. For example, RB/FAFF denotes the ratio of solving times of the options RB and FAFF. We have also reported the number of extra unit clauses added (due to more subgoals) as a percentage of the number of clauses in the encoding of original problem. The results in Fig. 5 are derived using results from Fig. 1,2,3 and 4. The number of extra unit clauses added can be found by subtracting the number of clauses in Fig. 1,2 and 3 from the number of clauses in Fig. 4.

Let us consider the results from Fig. 4. bw_large.b1 was solved by options RFFA and RB in times closer to those for other options. Results from Fig. 2 show that neither RFFA nor RB option could solve this problem within 1 hour. This shows that the extra unit clauses had a significant impact on the solving times of these options. The ratio of solving times of the options RAFF and FABF on robot.a reduced from 22.54 to 3.25 after extra subgoals were added. RAFF and FABF had respectively highest and lowest solving times on the original robot.a problem. The number of extra clauses added was 0.242 % of the number of clauses in the original encoding. The ratio of solving times of the options RFFA and BABF on m1_logistics.a went down from 35.56 to 1.53 when extra clauses (0.078 %) were added. RFFA and BABF had respectively highest and lowest solving times on the original problem. The ratio of solving times of RB and FAFF options on bw_large.a went down from 357.27 to 0.997 after addition of 84 unit clauses to the encoding. These were just 0.016 % of the number of clauses in the original encoding. RB and FAFF had respectively highest and lowest solving times on the original bw_large.a problem. The ratio of solving times of BFRA and BAFF on m2_logistics.a went down from 10.56 to 0.17 after the addition of 72 unit clauses (0.08 % of the number of clauses in the original encoding). BAFF option had the lowest solving time on the original problem.

5 Discussion

To evaluate the effect of directional solving on SAT encodings, we developed the directional solver DSatz which provides 21 directional solving options and conducted its empirical evaluation. These options can be viewed as various variable ordering heuristics. It is computationally very cheap to apply these. Though variable ordering heuristics have been widely investigated and shown to be useful in constraint satisfaction and SAT solving, their potential had remained under-investigated in planning as satisfiability.

With the exception of [4], there is no work on solving SAT encodings using variable ordering heuristics. The implementation [4] is based on the idea of assigning values to action variables before fluent variables, and directionality is not necessarily followed. The SAT solver that they modified to demonstrate the importance of this variable ordering is Tableau.

Our results on bidirectional search on SAT encodings show that evaluating this on planning graph of Graphplan is worthy of investigation. This is especially more useful because larger problems generally have larger encoding sizes. These encodings do not always undergo a significant simplification. The current SAT solvers cannot efficiently handle these encodings, despite the very significant progress in SAT solving in the last decade. Thus it is difficult to fully exploit the potential of directional solving on SAT representation, in case of larger problems. Given that local search-based SAT solvers (that do hill climbing and can make random moves) [5] have been very successful, it will be worth investigating if integrating non-directional and directional solving is more efficient.

6 Conclusion

To evaluate the effect of directional solving on SAT encodings, we developed DSatz, a variant of the efficient systematic solver Satz. DSatz allows 21 options for solving SAT encodings, where values are assigned to action and/or fluent variables in forward and/or backward directions in/without intermittent manner. These can be viewed as computationally very cheap variable ordering heuristics. This allows a richer evaluation of SAT planning. We showed that directionality does matter in solving of SAT encodings of planning. We showed that the superiority of backward search and inferiority of bidirectional search in refinement planning do not show up in SAT planning. We showed that backward search on SAT encodings yields inferior performance than both forward and bidirectional searches. In fact the solving option RB that closely resembles the operation of a backward state-space refinement planner in SAT representations yields worst performance. The bidirectional solving option BABF and the solving option FARF yield better performance than other solving options, showing that the role of these searches in improving efficiency of other disjunctive planners like Graphplan is worthy of investigation.

Acknowledgement: This work was funded by NSF grant IIS-0119630 to Amol Mali. The authors thank Enrico Giunchiglia for useful comments.

References

[1] Fahiem Bacchus, Notes of the constraint satisfaction course offered at University of Toronto, in computer science department, Available online, 2000.

Problem	# Variables	# Clauses	FAFF	FARF	FABF
robot.a	4830	84695	37.21	33.99	33.71 (*)
robot.b	577	3381	15.48	14.95	14.73
robot.c	612	3591	50.61	51.45	50.41
robot.d	542	3171	61.4	58.79	57.77

Problem	RAFF	BAFF	BARF	BABF	FFFA
robot.a	759.81 (**)	38.72	34.45	38.69	718.22
robot.b	18.73	27.37	26.52	27.56	9.71 (*)
robot.c	63.08	91.93	89.06	92.35	8.88 (*)
robot.d	78.37	91.02	87.92	91.65	45.35

Problem	FFRA	FFBA	RFFA	BFFA	BFRA
robot.a	661.31	679.97	445.76	54.56	52.54
robot.b	10.4	10.43	128.92	14.02	14.5
robot.c	9.83	9.7	231.31	36.43	37.46
robot.d	46.31	46.61	66.76	50.15	50.04

Problem	BFBA	FB	RB	BB	Satz
robot.a	57.13	37.7	490.94	35.43	240.53
robot.b	14.06	51.18	122.65 (**)	65.16	29.78
robot.c	36.73	182.35	217.59 (**)	124.52	93.38
robot.d	55.49	103.89 (**)	68.56	43.9 (*)	48.89

Figure 3. Evaluation of directional solving of SAT encodings.

[2] Blai Bonet, Gabor Loerincs and Hector Geffner, A robust and fast action selection mechanism for planning, Proceedings of National Conference on Artificial Intelligence (AAAI), 1997, pp. 714-719.

[3] Blai Bonet and Hector Geffner, Planning as heuristic search: New results, Proceedings of European Conference on Planning (ECP), 1999.

[4] Enrico Giunchiglia, Alessandro Massarotto and Roberto Sebastiani, Act and the Rest will follow: Exploiting determinism in planning as satisfiability, Proceedings of National Conference on Artificial Intelligence (AAAI), 1998, 948-953.

[5] Henry Kautz and Bart Selman, Pushing the envelope: Planning, propositional logic and stochastic search, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1996.

[6] Henry Kautz, David McAllester and Bart Selman, Encoding plans in propositional logic, Proceedings of Knowledge Representation and Reasoning conference (KR), 1996.

[7] Henry Kautz and Bart Selman, Unifying graph-based and SAT-based planning, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1999.

[8] Chu Min Li and Anbulagan, Heuristics based on unit propagation for satisfiability problems, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997.

[9] Drew McDermott, Using regression graphs to control search in planning, *Artificial Intelligence*, 109(1-2): 111-160, 1999.

[10] Steven Minton, Learning effective search control knowledge: An explanation-based approach, Ph.D thesis, Technical Report CMU-CS-88-133, 1988.

[11] Biplav Srivastava, XuanLong Nguyen, Subbarao Kambhampati, Minh B. Do, Ullas Nambiar, Zaiqing Nie, Romeo Nigenda and Terry Zimmerman, ALTALT: Combining Graphplan and heuristic state search, AI Magazine, Fall 2001, pp. 88-90.

[12] Manuela M. Veloso, Nonlinear problem solving using intelligent causal commitment, Technical Report CMU-CS-89-210, 1989.

[13] Manuela M. Veloso and Peter Stone, FLECS: Planning with a flexible commitment strategy, *Journal of Artificial Intelligence Research*, 3:25-52, 1995

[14] Vincent Vidal and Pierre Regnier, Total order planning is more efficient than we thought, *Procs. of AAAI*, 1999, pp. 591-596.

Problem	# Variables	# Clauses	FAFF	FARF	FABF
bw_large.a	4518	520956	42.97	42.82	43.71 (**)
bw_large.b1	8190	1209420	129.64	129.15	129.21
robot.a	4830	84900	9.51	9.42	10.57
m1_logistics.a	4926	79086	5.59	1.18	1.15 (*)
m2_logistics.a	5586	89646	44.79 (*)	57.06	58.47

Problem	RAFF	RARF	RABF	BAFF	BARF
bw_large.a	43.08	42.79	42.84	42.62	42.86
bw_large.b1	129.24	129.36	129.01	129.2	129.28
robot.a	34.4	36.08	37.26	9.24 (*)	13.12
m1_logistics.a	315.39	367.06	370.32	4.55	5.44
m2_logistics.a	> 12 hrs	> 12 hrs	> 12 hrs	1166.83	771.8

Problem	BABF	FFFA	FFRA	FFBA	RFFA
bw_large.a	42.85	43.19	42.76	42.65	43.04
bw_large.b1	128.92	129.01	128.73 (*)	129.28	132.66
robot.a	10.01	18.6	19.05	19.1	36.55
m1_logistics.a	4.41	10.58	10.61	11.51	6.73
m2_logistics.a	1185.85	216.41	206.44	227.34	> 12 hrs

Problem	RFRA	RFBA	BFFA	BFRA	BFBA
bw_large.a	42.76	43.07	42.74	42.78	42.91
bw_large.b1	130.23	130.47	129.91	129.71	129.74
robot.a	38.38	38.56	17.28	17	17.89
m1_logistics.a	6.57	7.18	11.7	11.5	12.74
m2_logistics.a	> 12 hrs	> 12 hrs	217.3	203.91	234.54

Problem	FB	RB	BB	Satz
bw_large.a	42.93	42.85	43.05	40.56 (*)
bw_large.b1	129.19	130.62	129.21	> 15 min. (**)
robot.a	9.6	40.73 (**)	33.49	30.1
m1_logistics.a	27.97	389.31 (**)	8.66	2.16
m2_logistics.a	156.83	> 12 hrs	432.94	47.93

Figure 4. Empirical evaluation with extra unit clauses added (due to stronger goal).

Problem	Goal	Stronger Goal	% New clauses added
bw_large.b1	> 7.97 (RB/FAFF)	1.007 (RB/FAFF)	0.0085
bw_large.a	350.96 (RFFA/FABF)	0.98 (RFFA/FABF)	0.016
robot.a	13.22 (RFFA/FABF)	3.46 (RFFA/FABF)	0.242
m1_logistics.a	23.39 (RFFA/FAFF)	1.2 (RFFA/FAFF)	0.078
m2_logistics.a	> 19.27 (BFBA/BAFF)	0.2 (BFBA/BAFF)	0.08

Figure 5. Some of the changes in ratios of solving times when extra unit clauses added (due to stronger goal).