To Encode or not to Encode – I: Linear Planning

Ronen I. Brafman
Department of Math and CS
Ben-Gurion University
Beer Sheva, Israel 84105
brafman@cs.bgu.ac.il

Holger H. Hoos
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada
hoos@cs.ubc.ca

Abstract

Stochastic local search (SLS) techniques are very effective in solving hard propositional satisfiability problems. This has lead to the popularity of the encode & solve paradigm in which different problems are encoded as propositional satisfiability problems to which SLS techniques are applied. In AI, planning is the main area in which this methodology is used. Yet, it seems plausible that SLS methods should perform better when applied to the original problem space whose structure they can exploit. As part of our attempts to validate this thesis, we experimented with LPSP, a planner that applies SLS techniques to the space of linear plans. LPSP outperforms SLS applied to encoded planning problems that enforce a similar linearity assumption because of its ability to exploit the special structure of planning problems. Additional experiments (reported in a longer version of this paper) conducted on the Hamiltonian circuit problem lend farther support to our thesis.

1 Introduction

Rapid improvement in the performance of stochastic local search (SLS) methods for solving propositional satisfiability (SAT) problems coupled with the naturalness with which many problems can be reduced to SAT problems has led to the popularity of an encode & solve approach to problem solving. In this three phase approach, the original problem is reduced into a sentence in propositional logic; a satisfying assignment for the sentence is searched for using state of the art SLS algorithms; and a solution, if found, is converted back into a solution to the original problem. This method has become increasingly influential in the area of classical AI planning, where it is known as the planning as satisfiability approach (PAS) [Kautz and Selman, 1992].

But is it not more effective to apply SLS methods directly to the original problem space? In this paper we report on our investigation into this issue in the context of planning problems. We describe the LPSP planner (<u>l</u>inear <u>p</u>lan-level <u>s</u>tochastic <u>p</u>lanner) which uses SLS on the space of linear plans to solve planning problems. We chose to explore the space of linear plans because we found it easier to formulate and program heuristics, metrics, and moves in this space. LPSP has a number of operators for moving in this space that enable it to discover plans in a small fraction of the number of steps required by SLS algorithms operating on the encoded planning problems. Some of its moves are reasonably natural when projected to the encoded problem. Others have no apparent equivalent, are motivated by the particular structure of planning problems, and are based on intuitions developed in classical work on partial-order planners (e.g., causal links, threats, producers, consumers, plan repair). Each move is considerably more expensive than moves made by, e.g., WALKSAT [Selman *et al.*, 1994], yet the overall performance is better than that of SAT based algorithms operating on encoded planning problems that share the linearity assumption.

Our planner is not competitive with planners such as BLACKBOX [Kautz and Selman, 1998] in domains in which parallel plans are much shorter than linear plans, e.g., the logistics domain.[1] Because the search space size is exponential in the plan length, LPSP is in a considerable disadvantage in such cases. However, it performs better than SAT based planners when the use of concurrency leads to little or no decrease in plan length. More generally, it does better than the SAT based approaches when we use Medic's [Ernst *et al.*, 1997] encodings in which the linearity assumption is enforced. We believe that this latter comparison is more valid when we come to examine the encoded vs. un-encoded formulations. In fact, we believe that these observations will apply when we extend this work to parallel plan structures. In Section 5, we discuss evidence from current research supporting this view. Overall, our results seem to support the thesis that SLS methods applied to problem domains with sufficient structure can outperform the encode & solve approach.

2 The Planning as Satisfiability Approach

Given a planning problem and some natural number n, we can generate a propositional formula whose models correspond to n step solutions to the problem. There are a number of schemes for generating such a formula (see, e.g., [Ernst *et al.*, 1997]), but the essential idea is that variables in this formula represent the value of the propositions describing the domain at different time points as well as the actions that are or are not applied at each time point. Hence, these variables contain information about the dynamics of the world throughout the execution of the plan. The formula contains constraints that

[1]By plan-length we refer to the number of time-steps in the plan representation. Parallel plan representations, unlike linear representations, allow the assignment of more than one (non-interfering) actions to a single time point.

these variables must satisfy if they describe the state of the world during the execution of a valid plan. From the value assigned to variables representing actions in any satisfying assignment we can easily infer a valid plan.

We can extend this idea into a planning algorithm as follows. Starting with some initial guess for n, we generate an appropriate formula. If it is satisfiable, we can get a plan by inspecting the satisfying assignment. Otherwise, we increase n and repeat.

As shown in [Kautz and Selman, 1996], this approach can be quite efficient thanks to the availability of fast SLS-based algorithms for SAT problems such as WALKSAT [Selman *et al.*, 1994]. WALKSAT starts by randomly picking a truth assignment. At each step, WALKSAT randomly chooses an unsatisfied clause c and makes it satisfied by flipping the value of one of its variables v. If possible, v is chosen so that no currently satisfied clause becomes unsatisfied by this flip. Otherwise, the variable is either chosen randomly or by selecting the flip that maximizes the number of satisfied truth assignments – called the assignment's *score*.

3 The LPSP Planner

Search space The LPSP planner performs SLS on a space of fixed length linear plans. As in the PAS approach, LPSP is given this length as part of its input. Hence, the states of the search space consist of sequences of ground actions of length n, where n is an input parameter. Null actions (i.e., actions with no preconditions or effects) are allowed as part of the sequence. In principle, with the introduction of appropriate operators, it is a simple matter to let LPSP operate on plans of varying length, much like conventional planners. However, we have not experimented with this option, yet.

Scoring plans LPSP scores (or more accurately, penalizes) plans by a weighted sum of plan flaws. A flaw arises in a plan when an action a (the *consumer*) has a precondition p and the closest predecessor a' of a that influences the value of p (the *cloberer*) has $\neg p$ as an effect. Each flaw's weight is d, the distance between a' and a. (In some domains, a weight of d^2 or d^0, performs better.) Each plan is penalized by the sum of the weights of all its flaws. Naturally, a plan is valid iff it has no flaws and consequently has 0 penalty.

Example 1: Consider a blocks' world domain with four blocks and actions of the form MOVE(x, y, z) which takes block x from y and places it on z. Its preconditions are ON(x, y), CLEAR(x), and CLEAR(z). Its effects are ON(x, z), CLEAR(y), $\neg$ON(x, z), $\neg$CLEAR(z).

Suppose that initially C is on A, and that B and D are on the table, and we want to reach a goal state in which A is on B and B is on C. Consider the following three step plan: MOVE(C,A,D), MOVE(B,D,C), and MOVE(A,TABLE,B). This plan contains a single flaw: the ON(B,D) precondition of MOVE(B,D,C). The closest predecessor of this action influencing ON(B,D) is the initial action,[2] which has $\neg$ON(B,D) as an effect. In this case $d = 2$ and so the plan's penalty is 2.

Moving around the search space Since plan space is searched, we can use various operators for moving in this search space that do not have natural counterparts in the space of truth assignments. In particular, the use of special plan reordering operators is crucial to LPSP's success. Without them, it usually does not find a solution.

(1) *Best-Replacement*: This operator scores all plans that differ from the current plan by a single action, i.e., plans in which one of the current plan's steps was replaced. It returns the plan with the least penalty score among these.[3] In Example 1, we would consider plans in which one of the current steps was replaced by some ground action. In particular, when we replace the MOVE(B,D,C) step with MOVE(B,TABLE,C) we get a plan with no flaws.

(2) *Flaw-Repair*: This operator randomly chooses a flaw from the current plan flaws. It then scores all possible plans in which this flaw is repaired by replacing one plan step between the cloberer and the consumer. The new action must establish the precondition of the consumer destroyed by the cloberer. If none of these plans has a lower penalty score than the current plan, it examines plans in which the consumer is replaced by some other action.

Flaw-repair is motivated by a similar strategy employed by WALKSAT where a "flawed" (i.e., unsatisfied) clause is chosen and repaired by changing the value of one of its variables.

Example 2: Consider actions that repair the flaw of the plan in Example 1, i.e., actions that have ON(B,D) as an effect. Such actions are of the form MOVE(B,?X,D) and the best scoring plan is obtained when we use MOVE(B,TABLE,D). The resulting plan will be: MOVE(B,TABLE,C), MOVE(B,D,C), and MOVE(A,TABLE,B). This plan has a single flaw: the CLEAR(A) precondition of the last action. The closest predecessor of the last action affecting CLEAR(A) is the initial action, and so this plan's penalty is 3, which is higher than the penalty of the original plan. Consequently, we consider actions that replace the consumer, namely the second step, among them MOVE(B,TABLE,C), which gives us a solution.

(3) *Reorder-1*: We generate a directed graph whose vertices are the current plan steps. An edge exists from plan step a to step a' if a (the *producer*) has an effect that is a precondition of a' (the consumer). First, we use this graph to throw out steps that are not useful (by replacing them with null actions). A step is *useful* if it has an effect that is part of the goal conjunct or an effect that is a precondition of a useful step. Adjacent plan steps, between which we have edges in both directions, are deleted as well. Now, we add additional edges to the graph that reflect a heuristic notion of *threats*: If a, a' are nodes in the graph between which there is no edge, and if a clobers some precondition of a' we say that a *threatens* a' and we add an edge from a' to a.

Finally, we attempt to generate an ordering consistent with the graph, i.e., such that step a will precede step a' if there is a path from a to a'. Usually, this is not possible, and we heuristically order the edges as follows: One of the nodes with the minimal number of incoming edges is chosen as the first step. It is removed from the graph, and the next step is chosen from this revised graph in the same fashion.

Example 3: We have two stacks of blocks: A on B on C and D on E on F; our goal state is F on B on E and A on C on D; plan length is 7. Suppose that our current plan is: (1) MOVE(A,B,TABLE), (2) MOVE(A,TABLE,C), (3) MOVE(E,F,TABLE), (4) MOVE(B,C,E), (5) MOVE(F,TABLE,B), (6) MOVE(D,E,C), (7) MOVE(C,TABLE,D).

[2]We use the well known technique of inserting a (fixed) fictitious initial action that "sets-up" the initial state as its effects and a fictitious final action that has the goal as its precondition.

[3]We break ties randomly.

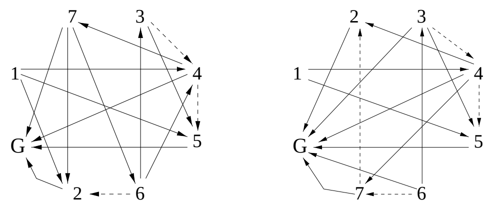

Figure 1: Graphs generated in Examples 3 (left) and 4 (right). Solid edges stem from producer-consumer relation, broken edges stem from threats.

The producer/consumer pairs appears in Figure 1 (left) and are denoted by solid edges. According to this graph, all nodes are useful. Next, we add edges reflecting threats, denoted by broken edges. Now, we start reordering the nodes. The only node without incoming arcs is (1), and it is assigned to the first plan step. Deleting it, we see four nodes with a single incoming edge (2),(3),(6),(7), among which we can choose arbitrarily. Suppose we chose (6). Removing it from the graph, we see that (3) has no incoming edges, and we choose it. Next comes (4) and then either (5) or (7). If we choose (7), we can next choose between (5) and (2). Hence, one possible resulting plan is: (1) MOVE(A,B,TABLE), (6) MOVE(D,E,C), (3) MOVE(E,F,TABLE), (4) MOVE(B,C,E), (7) MOVE(C,TABLE,D), (5) MOVE(F,TABLE,B), (2) MOVE(A,TABLE,C).

Example 4: Here is a similar plan in which the correct plan steps are ordered incorrectly. (1) MOVE(A,B,TABLE), (2) MOVE(A,TABLE,C), (3) MOVE(E,F,TABLE), (4) MOVE(B, C, E), (5) MOVE(F, Table, B), (6) MOVE(D, E, TABLE), (7) MOVE(C, TABLE, D). The initial graph contains the solid edges in Figure 1 (right). All nodes are useful. When we add threat arcs we obtain the additional broken edges. Nodes (1) and (6) have no incoming edges. Either choice is correct. We choose (1). (6) is still the only node without incoming edges, and we choose it next. Node (3) comes next, followed by (4). Now we can choose between (5) and (7). Both choices are correct. If we choose (7) we can then choose (2) as well. Whichever choice we made, we would get a correct plan.

(4) *Reorder-2*: Reorder-2 attempts to partition the set of actions into disjoint ranks such that, roughly, each rank is preceded by a rank containing actions that supply its preconditions. The actions within a rank are not needed by each other. After removing unneeded actions, we attempt to reorder actions within a rank based on whether they destroy each others' effects. A more detailed description is deferred to the full version of this paper

(5) *Reorder*: Applies Reorder-1 with probability 0.5 and otherwise applies Reorder-2.

Initialization We initialize the plan by performing sto/-chastic bi-directional search. That is, if n is the plan size, we run a standard regression algorithm in which branches are chosen stochastically for $n/2$ steps. This determines the latter half of the plan. Then, we run stochastic forward search for $n/2$ steps, which generates the first half of the plan.

Reachability Analysis Before we start the planning process, we perform a fast heuristic reachability analysis in order to prune the number of actions considered at each time step. First, we determine which actions are applicable at the initial state. Let E_1 be the set of effects of all these actions. Clearly, actions that have a precondition outside E_1 can be pruned as candidates for the second plan step. We continue, letting E_k be the effects of all actions that were not pruned as candidates for the $k + 1$ time step, etc.

The LPSP algorithm The algorithm is described below; upper-case names are constants.

```
Procedure LPSP(MAX_TRIES,MAX_STEPS):
  Perform reachability analysis;
  Repeat for MAX_TRIES or until solution found:
    Initialize Current_Plan;
    Repeat for MAX_STEPS or until solution found:
      If (Current Step MOD 5 = 0)
        Obtain New_Plan by applying Best-Replacement;
      Else
        Obtain New_Plan by applying Flaw-Repair;
      End If;
      If solution not found
        If (New_Plan's penalty score <
            (Current_Plan's penalty score + DOWN_THRESH))
          Current_Plan := New_Plan;
        Else
          If Almost_Solution holds
            Threshold := GREEDY_IF_ALMOST;
          Else
            Threshold := GREEDY_IF_NOT_ALMOST;
          End If;
          With probability Threshold:
            Current_Plan := New_Plan;
          With probability (1-Threshold):
            If (Almost_Solution or (New_Plan = Current_Plan))
              Apply Reorder;
            Else
              New_Plan := Current_Plan;
            End If;
        End If;
      End If;
    End Repeat;
  End Repeat;
```

Notes: Almost_Solution holds if New_Plans's penalty score is lower than OPT_THRESHOLD, and we did not call Reorder in the previous step. We keep a tabu list of pairs (whose length is set to 2 in our experiments) containing the last two actions added and their respective positions. We disallow new plans in which an action is inserted in a position for which the corresponding action-position pair appear in the tabu list.

State of the planner LPSP is implemented in C++. It does not have a domain parser, yet. Hence, for each planning problem we write a procedure that specifies the propositions that hold in the initial and goal state; and for each action schema we define a class whose parameters are the schema's parameters and a list of the appropriate preconditions and effects. In addition, we supply a procedure that, given an action identifier (which is simply some integer), returns an action object. These operations have negligible running times and we believe that the time required to parse a domain description into a more generic action representation will be inconsequential (and, most likely, smaller than the time required to encode the same problem into SAT).

4 Experimental Results

We empirically compared LPSP to the PAS approach on problem instances from three well-known planning domains: the blocks world and logistics domains [Kautz and Selman, 1996; Ernst *et al.*, 1997] and the artificial D1S1 domain [Barrett and Weld, 1994].[4] For the PAS approach, the performance depends on three components: (i) the SAT-encoding of the planning instances, (ii) polynomial simplification algorithms which are applied to the SAT formula before general SAT solvers are invoked, and (iii) the SAT algorithm used to solve the simplified encoded problem instance. All three components have a significant influence on the overall performance; in particular, the solver performance heavily depends on the encoding and simplification algorithms used to create the SAT instance. Considering that additionally the SAT algorithms have several parameters which have to be tuned to achieve optimal performance, the number of choices over which one would have to optimize in order to get the best possible performance for the PAS approach is too high to practically allow an exhaustive analysis. Therefore, for this comparative study, we chose the following approach:

- We used two linear SAT encodings which are known to be good from the literature; for the blocks world domain, this is the linear encoding with operator splitting described in [Kautz and Selman, 1996], for the logistics and D1S1 we used the "erse" (explanatory frame axioms, regular operator splitting, sequential plans, full type elimination) encoding as described in [Ernst *et al.*, 1997].
- We considered several simplification strategies, as provided by Jimmy Crawford's COMPACT simplifier; besides "no simplification" [-] these were unit propagation and pure literal elimination [p] plus the unary [u] and binary [b] failed literal strategies which are based on efficiently computing unary and binary implicates. For the blocks world domain, we also considered the simplifier based on unit propagation and subsumption [ps] which is part of the SATPLAN system.
- We focussed on stochastic local search algorithms for SAT; in particular, we considered the four best-performing WALKSAT variants described in [McAllester *et al.*, 1997] which are among the fastest existing SAT solvers. To get an impression, how LPSP's performance compares to that of systematic SAT algorithms, we also used SATZ [Li and Anbulagan, 1997], one of the best deterministic SAT solvers.

For the stochastic solvers, we took great care to optimize the parameters (especially the noise parameter which is known to significantly affect performance) so that the overall time required for finding a solution was minimized. This was done for each SAT instance independently, ensuring that for our comparison we got an approximately optimal performance. Because for a given problem instance, the different types of local search steps in LPSP require a variable amount of time, and for the PAS approach, the simplification and solving time have to be taken into account, we compared CPU times as measured on a PC with a 400MHz Pentium II processor and 256 MB RAM running under Linux (Red Hat 5.2). For both LPSP and the stochastic SAT solvers, our comparison is based on 100 tries, using MAX_STEPS settings high enough to guarantee that a solution was found in all tries. For LPSP, we left the value of many parameters fixed throughout the experiments (e.g, tabu list length was 2 and DOWN_THRESH was 5) and varied the values of GREEDY_IF_ALMOST, GREEDY_IF_NOT_ALMOST, and OPT_THRESH.

The problem instances have the following characteristics:

- Blocks World: We used the instances described in [Kautz and Selman, 1996]; these are: `bw_large.a` (9 blocks, 6 steps minimal linear plan length), `bw_large.b` (11 blocks, 9 steps), `bw_large.c` (15 blocks, 14 step plan), and `bw_large.d` (19 blocks, 18 step plan).
- Logistics: We used three instances in which there were 8 packages and 3 cities but only a small number of packages change their location. The instances are `log.new.a` (6 steps minimal linear plan length), `log.new.b` (10 steps), and `log.new.c` (16 steps).
- D1S1: We used domain sizes $n = 15$ and 30 and four instances per domain size: `d1s1-`n`.a` (15/30 steps minimal linear plan length), `d1s1.b` (12/23 steps), `d1s1.c` (8/16), and `d1s1.d` (14/28 steps).

In Table 1, we report our experimental results. For LPSP, we report the mean and median CPU times over 100 successful tries (see above) as well as the variation coefficient (standard deviation / mean), which gives a scaling independent impression of the variability of the observed run-times. For PAS/WALKSAT, we report the CPU times for simplifying the formula and the mean CPU time required by WALKSAT for solving the resulting simplified problem instance averaged over 100 successful tries. Furthermore, we report the variation coefficient for the time used by WALKSAT.[5] Note that here, we report only the best results we found for any simplification and WALKSAT combination, using approximately optimal noise parameter settings for WALKSAT.[6] For PAS/SATZ, as only deterministic algorithms are used, we report the CPU times for simplification and SATZ.

As can be seen from Table 1, LPSP shows superior performance compared to PAS/WALKSAT as well as PAS/SATZ on almost all problem instances. For the blocks world instances, LPSP is between 3 and 13 times faster than PAS/WALKSAT. Interestingly, for PAS/WALKSAT, the time required for simplifying the formula dominates the overall performance. Since different from WALKSAT, the implementations of the simplification algorithms are not optimized for speed, one might expect a significant reduction in overall performance when using optimized simplifiers. However, for the larger instances, LPSP shows superior performance even when (unrealistically) assuming that simplification would come for free. Furthermore, LPSP requires much less tweaking then WALKSAT, for which various simplification strategies must be considered, various heuristics, and various parameter tuning per heuristic. In addition, our LPSP implementation is certainly not fully optimized.

[4]The first two were chosen because of their use in the original SATPLAN paper, and the third because [Kautz and Selman, 1996] mentions that it seems difficult for SATPLAN.

[5]Our actual experimental methodology is based on measuring run-time distributions (RTDs) as outlined in [Hoos and Stützle, 1998]; because of the limited space the RTDs could not be reported here, but this data is available from the authors.

[6]The best-performing WALKSAT variants were TABU for the Blocks World instances and Novelty for all others. The approximately optimal noise settings varied between the instances and seemed to be strongly dependent on the simplification strategies applied before.

instance	LPSP		PAS/WALKSAT		PAS/SATZ
	mean	vc	mean	vc	
bw_large.a	0.096	0.53	0.73+0 [u]	n/a	0+0.56 [-]
bw_large.b	1.2167	0.82	2.78+0.8250 [u]	0.88	0+1.44 [-]
bw_large.c	4.6219	0.61	47.62+15.337 [ps+u]	0.99	0+5.82 [-]
bw_large.d	31.1764	0.60	101.14+50.912 [ps]	0.94	*
log.a	0.068	0.48	4.33+< 0.01 [u]	n/a	4.33+< 0.01 [u]
log.b	0.6881	1.37	11.56+< 0.01 [b]	n/a	11.56+0.015 [b]
log.c	13.5037	1.11	1070.22+4.9477 [b]	0.67	*
d1s1-15.a	0.0199	0.79	0.21+0 [p]	n/a	0+0.070 [-]
d1s1-15.b	0.0084	0.99	0+0.0123 [-]	0.61	0+0.080 [-]
d1s1-15.c	0.0056	1.14	0+0.0068 [-]	0.51	0+0.050 [-]
d1s1-15.d	0.0135	0.40	0.28+< 0.01 [u]	n/a	0+0.090 [-]
d1s1-30.a	9.68	0.34	0.90+0 [p]	n/a	0+0.59 [-]
d1s1-30.b	0.056	0.30	0+0.072 [-]	0.62	*
d1s1-30.c	0.023	0.22	0+0.033 [-]	0.33	0+0.43 [-]
d1s1-30.d	2.62	0.57	2.56+0.53 [u]	0.75	2.56+0.34 [u]

Table 1: Performance of LPSP versus PAS using the best simplification and solver variants with approximately optimal parameters. For stochastic solvers, we report the mean run-times over 100 successful tries and the variation coefficient (vc = standard deviation/mean). For both PAS approaches, we report the times required for simplifying (first number) and solving (second number) separately and also indicate which simplification method has been applied (see text for explanation) All run-times reported in CPU seconds; "*" indicates cases in which no solution was found in 60 CPU seconds (for SLS algorithms, 10 tries á 60 seconds were performed). For further details, see text.

The more complex SAT-simplifications seem to be far more effective for the logistics domain than for the blocks world or D1S1 problems. Here, the formulae obtained by simplifying with the "unary/binary failed clauses" options are almost trivial for WALKSAT. However, although they have polynomial worst-case complexity, these simplifications are quite expensive in practice. So here, again, for PAS/WALKSAT the overall performance is dominated by the time required for simplification. Comparing the overall performance, LPSP is between 15 and 80 times faster than PAS/WALKSAT. Even assuming that the implementation of the simplifier could be considerably more optimized than LPSP, it is hard to imagine that PAS/WALKSAT could reach LPSP's performance on this domain. It should be noted, however, that for the logistics domain, most problem instances have significantly shorter parallel plans which are much easier to find than sequential plans. But since LPSP is a linear planner, it would be unfair to compare its performance to PAS approaches which allow for parallel plans. Finally, even for D1S1, a domain which has been noted to be difficult for SATPLAN using SLS for solving the SAT instances [Kautz and Selman, 1996], LPSP shows better performance than PAS/WALKSAT. The only exception is instance d1s1-30.a, which is solved by simplification (unit propagation) alone. The reason for this is that the d1s1-n.a instances have a chaining structure of the goals and operators which is very regular, but, in our opinion, highly untypical for realistic planning problems. We also believe that LPSP's performance on this domain can be significantly improved by optimizing its parameter settings.

Regarding the results on PAS/SATZ, LPSP's performance is still considerably better on all instances except for instance d1s1-30.a, which in principle is solved by simplification (unit propagation) alone. Nevertheless, for this particular instance, SATZ is actually faster then the simplification algorithm we use, probably because it is more efficiently implemented. Interestingly, when compared to PAS/WALKSAT, PAS/SATZ shows a better performance for 7 of our 11 benchmark instances. For the biggest blocks world and logistics instances, as well as for one of the d1s1-30 instances, SATZ did not find a solution in 60 seconds — which could possibly indicate that SATZ does not scale as well as SLS approaches (LPSP and PAS/WALKSAT). However, this issue needs to be further investigated, especially using newer, randomized systematic search algorithms [Gomes *et al.*, 1998].

5 Discussion

The thesis that SLS-based algorithms operating on unencoded problems can fare better than those operating on encoded problems cannot be proved experimentally. The performance of LPSP does, however, lend it greater credibility, especially when we compare the effort invested in it with that invested in work on SAT-based SLS algorithms in general and the PAS approach specifically.

In a longer version of this paper, we describe experiments which we conducted on two additional problem domains: Hamiltonian circuit and binary CSPs. The first domain has interesting, inherent structure, and we have been able to obtain better results using an SLS-based algorithm applied to the graph description directly. The second problem domain is more general and closely resembles SAT problems. There, we found that SAT-based methods perform better than the SLS-based algorithms we experimented with.[7]

Of course, one could argue that in the area of planning in general, SLS-based methods applied to encoded problems that do not enforce the linearity assumption have the upper hand. However, recent research is starting to show that the situation is a bit more complicated. [Serina and Gerevini, 1998] employ SLS methods to search Graphplan's planning graph. Their algorithm outperforms PAS based methods on some problems, and in particular, those in which concurrent

[7]See also [Hoos, 1999].

plans are much shorter. Our algorithm seems to outperform PAS based approaches when concurrency is not an important factor (e.g., the blocks' world instances). The best SAT-based planner, BLACKBOX [Kautz and Selman, 1998], relies more on traditional plan representations than the original SATPLAN planner [Kautz and Selman, 1996]. BLACKBOX goes very far with the original planning problem before converting it into a propositional formula: It first generates a planning graph and encodes this graph together with the mutual exclusion constraints it generated. More importantly, recent work [Giunchiglia *et al.*, 1998] shows that better performance can be obtained if the variables describing state properties are treated as dependent variables (i.e., one chooses an assignment for the action variables and this determines the value of the state variables). What this result mean is that it is better to search over the space of assignments to actions than over the whole space of truth assignments. The space of action assignments is very similar to LPSP's search space.

Naturally, we have not heard the last word in SAT algorithms, and recent work on exploiting variable dependences [Kautz *et al.*, 1997] may allow us to utilize knowledge about problem structure in solving the encoded problems. Another promising direction in which little work has been carried out is the analysis of the encoded problems for useful features of search space structures (see also [Hoos, 1999]) or syntactic properties. Such features may stem from deep properties of the encoded problem. To date, most SAT solvers do not attempt to utilize such information (although it could be argued that formula simplification routines do something of this sort). But perhaps there are algorithms that can operate much better than the standard methods on sentences with special properties. This is particularly interesting if such properties are shared by all problems from a particular domain. If this turns out to be the case, then, indeed, the encode & solve approach may be our best bet.[8]

To use Chapman's terminology, LPSP is a somewhat "scruffy" planner. It would be nice to see a simpler, more principled and general planner based on SLS techniques. However, it is already much easier to optimize than the PAS based algorithms that have three distinct phases (translate, simplify, solve) each of which needs to be fine-tuned separately. Moreover, it may be the case that good SLS algorithms must be a bit involved if they are to exploit domain structure.

LPSP's reordering operators are interesting in their own right and could possibly be integrated into plan reuse and plan repair algorithms. Plan reordering has been considered in the past (see [Backstrom, 1998] and references therein) but typically with the goal of optimizing the order of correct plans. The idea of repairing plans in general is not new either, and goes back at least to [Sussman, 1975]. The heuristic, rather than more systematic, approach LPSP takes stems from the need to balance the desire for a better, and more sophisticated order analysis (which is likely to be intractable), and the need to maintain a low cost for each step. Finally, we note that different types of domain specific knowledge can be used to focus the search process, e.g., by using general variables rather than binary variable to specify state information, by restricting the type of operators that should be considered in various situations, etc.

[8]See [Brafman, 1999] for a recent attempt along this direction.

Acknowledgments

Craig Boutilier was instrumental in the development of earlier versions of LPSP. The first author is partially supported by the Paul Ivanier Center for Robotics Research and Production Management.

References

[Backstrom, 1998] C. Backstrom. Computational aspects of reordering plans. *Journal of AI Research*, 9:99–137, 1998.

[Barrett and Weld, 1994] A. Barrett and D. Weld. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[Brafman, 1999] R. I. Brafman. Reachability, relevance, resolution, and the planning as satisfiability approach. In *Proc. of IJCAI'99*.

[Ernst *et al.*, 1997] M. D. Ernst, T. D. Millstein, , and D. S. Weld. Automatic SAT-compilation of planning problems. In *Proc. IJCAI'97*, 1997.

[Giunchiglia *et al.*, 1998] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proc. 15th Nat. Con. on AI*, 1998.

[Gomes *et al.*, 1998] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI'98*, 1998.

[Hoos, 1999] H. H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proc. IJCAI'99*, 1999.

[Hoos and Stützle, 1998] H. H. Hoos and T. Stützle. Evaluating Las Vegas algorithms — pitfalls and remedies. In *Proc. UAI'98*, 1998.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. 10th ECAI*, 1992.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. 13th NCAI* , pages 1194–1201, 1996.

[Kautz and Selman, 1998] H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search*, 1998.

[Kautz *et al.*, 1997] H. Kautz, D. McAllester, and B. Selman. Exploiting variable dependency in local search. 1997.

[Li and Anbulagan, 1997] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. IJCAI-97*, 1997.

[McAllester *et al.*, 1997] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *AAAI'97*, pages 321–326, 1997.

[Selman *et al.*, 1994] Bart Selman, Henry A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, volume 1, pages 337–343, 1994.

[Serina and Gerevini, 1998] I. Serina and A. Gerevini. Local search techniques for planning graphs. In *Proc. 17th UK Planning and Scheduling WS*, pages 157–168, 1998.

[Sussman, 1975] G. J. Sussman. *A Computer Model of Skill Acquisition*. MIT Press, 1975.