Using Satisfiability for Non-Optimal Temporal Planning

Masood Feyzbakhsh Rankooh

Advisor: Dr. Gholamreza Ghassem-Sani

Abstract

Using satisfiability for non-optimal temporal planning has not been investigated so far. The main difficulty in using satisfiability in temporal planning is the representation of time, which is a continuous concept. Previously introduced SAT-based temporal planners employed an explicit representation of time in the SAT formulation, which made the formulation too large for even very small problems. To overcome this problem, we introduce a novel method for converting temporal problems into a SAT encoding. We show how the size of the encoding can be reduced by abstracting out durations of planning actions. We also show that the new formulation is powerful enough to encode fully concurrent plans. We first use an off-the-shelf SAT solver to extract an abstract initial plan out of the new encoding. We then add the durations of actions to a relaxed version of the initial plan and verify the resulting temporally constrained plan by testing consistency of a certain related Simple Temporal Problem (STP). In the case of an inconsistency, a negative cycle within the corresponding Simple Temporal Network (STN) is detected and encoded into the SAT formulation to prevent the SAT solver from reproducing plans with similar cycles. This process is repeated until a valid temporal plan will be achieved.

Introduction

While satisfiability checking is a major approach in dealing with classical planning, it has been insufficiently exploited in the field of temporal planning. In fact, the only published SAT-based temporal planners are STEP [1] and T-SATPLAN[2], which are both optimal planners. Besides, none of well-developed techniques for increasing the speed of SAT-based planners have been used in temporal planning.

Both STEP and T-SATPLAN use an explicit representation of time in their encodings. Generally speaking, in these SAT-based approaches, layer i is exactly one time unit ahead of layer $i+1$. If an action has a duration of d and its starting point is in layer i, then its ending point is restricted to be in layer $i+d$. Such a representation makes an enormous number of layers to be present in the SAT formulation while not being used in the final plan. This problem worsens as the duration ratio between the most durative action and the least one gets higher. Encoding more layers increases the size of the produced formula and may cause the planner to become highly memory sensitive. Even if the planner is not running out of memory, a linearly larger number of variables means an exponentially larger search space for finding a solution and as a result, a substantial decrease in the speed of the planner.

In this report, we explain an alternative encoding approach in which instead of using an explicit representation of time, layers are used merely for determining the order of actions. In fact, in our representation, layer i is ahead of layer $i+1$, but the actual difference in their times is not fixed. Therefore, an action can have its start in layer i, and its end in layer $i+1$, regardless of its duration. In other words, the durations of actions are abstracted out at the time of producing the SAT encoding.

Abstracting the durations of actions is not a new concept in the field of temporal planning. Some temporal planners have used such abstractions for guaranteeing the completeness in an important subset of temporal problems, called problems with required concurrency [3]. CRIKEY [4] was the first planner that separated the planning and scheduling processes by eliminating all durations in the planning phase. CRIKEY tests the validity of the resulting plan later in the scheduling phase, by solving a simple temporal problem. POPF [5], a descendant of CRIKEY, uses a linear programming method in its scheduling phase. Both POPF and CRIKEY take benefit from an enforced hill-climbing search method [6] in addition to standard best-first search mechanism. POPF is regarded as the state-of-the-art planner for solving the temporal problems with required concurrency.

The block diagram of our approach is depicted in figure 1. Similar to CRIKEY and POPF, our planner, named ITSAT (Implicit Time SAT planner), will need a scheduling step, when an abstract plan is produced by the SAT solver. The durations should be given back to the actions and an exact time value should be assigned to each of them by solving a simple temporal problem. However, such a consistent temporal assignment may not exist. As we show later, the reason of the inconsistency can be easily identified as negative cycles in the graph representation of the corresponding STP [7]. We then, add certain variables and clauses to the SAT formula in order to prevent the SAT solver from reproducing such cycles.

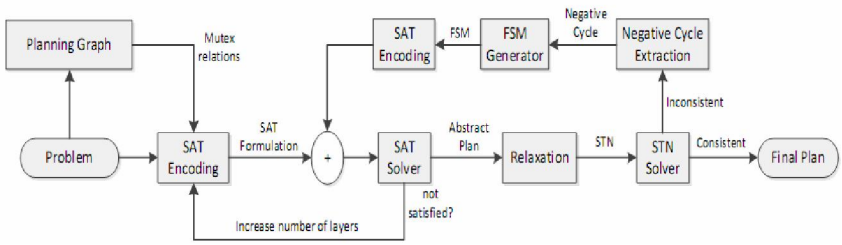

Fig. 1. The block diagram of ITSAT

Temporal Planning

We now give a formal description of temporal planning. Before that, we need a formal definition of a classical action. The definitions presented here are compatible with PDDL2.1 [8] where temporal actions can have separate preconditions and effects upon starting or ending.

Definition 1. A classical action o is a triple ($pre(o)$, $add(o)$, $del(o)$). All $pre(o)$, $add(o)$, and $del(o)$ are sets of facts and each fact is an atomic proposition. The set $pre(o)$ includes all the preconditions of o. The sets $add(o)$ and $del(o)$ contain the add and delete effects of o, respectively.

Definition 2. A temporal action a is defined by a positive rational duration $d(a)$, a starting event a^s, an ending event a^e, and an over-all condition $over(a)$. Each event is a classical action and the over-all condition is defined as a classical precondition. We also have: $action(a^s)=action(a^e)=a$. Action a is applicable in time t, if $pre(a^s)$ and $pre(a^e)$ are respectively held in time t and $t+d(a)$; and furthermore, $over(a)$ is held in open interval $(t,t+d(a))$. The application of a will cause the effects of a^s and a^e to take place in time t and $t+d(a)$, respectively.

Definition 3. A temporal planning problem is a quadruple $P=(A,F,I,G)$, where A is a set of temporal actions, F is the set of all the given facts, I is a classical initial state, and G is a set of classical goal conditions.

Definition 4. Suppose that $P=(A,F,I,G)$ is a temporal planning problem. A plan is represented by $\pi=\{(a_1,t_1),\ldots,(a_n,t_n)\}$, which means that action a_i is performed in time t_i.

SAT Encoding

We now explain how a temporal planning problem $P=(A,F,I,G)$ is encoded into its corresponding SAT formula. Suppose that E is the set of all starting and ending events of P, and we are constructing a formula with n conceptual layers. Variables and clauses that are essential for the soundness and completeness of the encoding are defined as follows.

We define some variables to represent all the facts of P.

- For every fact $f \in F$, and every $1 \le i \le n$, a variable $X_{f,i}$ is defined. Assigning 1 to $X_{f,i}$ means that the fact f is true in the conceptual layer i.

Although previous SAT-based temporal planners, STEP and T-SATPLAN, encode each temporal action with a single variable, our encoding has two variables for the starting and ending events of each action, and another variable for representing the situations when the action is still running i. e., it is open.

- For every event $e \in E$, and every $1 \le i < n$, a variable $Y_{e,i}$ is defined. Assigning 1 to $Y_{e,i}$ means that the event e is performed in the conceptual layer i.
- For every action $a \in A$, and every $1 \le i < n$, a variable $W_{a,i}$ is defined. Assigning 1 to $W_{a,i}$ means that the action a is open (started but has not yet finished) in the conceptual layer i .

We also need some clauses for constraining the values of the variables of the first and the last layers in an appropriate way.

- For every fact $f \in I$, a clause $X_{f,1}$ is asserted to ensure the presence of initial facts in layer 1.
- For every fact $f \in F\text{-}I$, a clause $\neg X_{f,1}$ is added to ensure that only the facts of initial state can be present at layer 1.
- For every fact $g \in G$, a clause $X_{g,n}$ is asserted to ensure the presence of all the goal conditions in the last layer.
- For every action $a \in A$, a clause $\neg W_{a,1}$ is asserted to ensure that no action is open in layer 1.
- For every action $a \in A$, a clause $W_{a,n} \rightarrow Y_{ae,n}$ is added to ensure that no action is remained unfinished in layer n.

We need certain clauses to handle the necessary conditions of performing an event in a layer and propagating the effects of that event to the next layer. The following clauses guarantee that if an event e is being performed in

layer i, then its preconditions are available in layer i, and its effects are held in layer i+1.

- For every event $e \in E$, every $f \in pre(e)$, and every $1 \le i < n$, a clause $Y_{e,i} \rightarrow X_{f,i}$ is added.
- For every event $e \in E$, every $f \in add(e)$, and every $1 \le i < n$, a clause $Y_{e,i} \rightarrow X_{f,i+1}$ is asserted.
- For every event $e \in E$, every $f \in del(e)$, and every $1 \le i < n$, a clause $Y_{e,i} \rightarrow \neg X_{f,i+1}$ is asserted.

Note that these constraints are not sufficient to make a valid plan. Many planning representations, including PDDL2.1 [8], forbid a proposition to be deleted by an event if it is needed by at least one other event at the same time. If the effects of each event take place only in the next layer, detecting such conflicts will be impossible. To overcome this flaw, quite similar to previous classical and temporal SAT-based planners, we use clauses that explicitly encode the mutual exclusion between conflicting events. As an example, if event a deletes a precondition of event b, then the clause $Y_{a,i} \rightarrow \neg Y_{b,i}$ is added to the formula for every i to ensure a and b will never be performed at the same layer.

Beside preconditions of an event, some other conditions must hold to enable that event to occur in a certain layer. We assume that two copies of the same ground action can never be concurrent. Even though PDDL2.1 allows this kind of concurrency, none of the domains we observed during our empirical analysis had such requirements.

- For every $1 \le i < n$ and every event $e \in E$, if e is the starting event of action a, a clause $Y_{e,i} \rightarrow \neg W_{a,i} \wedge W_{a,i+1}$ is added to ensure that two copies of action a will not be concurrent. This clause also assures us that if a is started in layer i, it will be considered as an open action in layer $i+1$.
- For every $1 \le i < n$ and every event $e \in E$, if e is the ending event of action a, a clause $Y_{e,i} \rightarrow W_{a,i} \wedge \neg W_{a,i+1}$ is added to ensure that only open actions can be ended; moreover, if an action is ended in layer i, it will no longer be considered as an open action in layer $i+1$.
- For every $1 \le i < n$, every action $a \in A$, and every fact $f \in over(a)$, we add a clause $W_{a,i} \rightarrow X_{f,i}$. This clause assures us that when an action is still open, its over-all conditions are maintained.

Finally, several kinds explanatory frame axioms are needed for preventing the values of the variables from changing without any reason.

- For every $1 \le i < n$ and every $f \in F$, we add a clause $\neg X_{f,i} \wedge X_{f,i+1} \rightarrow (\bigvee_{f \in add(e)} Y_{e,i})$ to prevent any fact from being arbitrarily asserted in layer i.
- For every $1 \le i < n$ and every $f \in F$, we add a clause $X_{f,i} \wedge \neg X_{f,i+1} \rightarrow (\bigvee_{f \in del(e)} Y_{e,i})$ to prevent any fact from being arbitrarily deleted from layer i.
- For every $1 \le i < n$ and every $a \in A$, we add a clause $\neg W_{a,i} \wedge W_{a,i+1} \rightarrow Y_{e,i}$, where e is the starting event of action a. This clause ensures that no action can become open in layer $i+1$, without having its starting event in layer i.
- For every $1 \le i < n$ and every $a \in A$, we add a clause $W_{a,i} \wedge \neg W_{a,i+1} \rightarrow Y_{e,i}$, where e is the ending event of action a. This clause ensures that if an action is open in layer i but not open in layer $i+1$, its ending event must be present in layer i.

It should be obvious from our SAT encoding that durations of actions play no role in the encoding. However, every other constraint of temporal planning is encoded into the SAT formulation. In other words, if we solve the SAT formula with a SAT solver, the resulting plan will be a valid temporal plan, unless there is an inconsistency among the durations of actions. We will discuss this issue in more details later.

The Scheduling Phase

Assume that we have solved a SAT formula produced by the encoding phase described is section 3 and obtained a solution for it. Let $e_1,\ldots,e_m$ be the events whose corresponding SAT variables have a value of 1 in the solution. Suppose that we have given different names to different occurrences of the same action in the solution, so all the events $e_1,\ldots,e_m$ are unique. Let $layer(e_i)$ be the number of layer in which e_i has occurred.

Events $e_1,\ldots,e_m$ must then be scheduled in order to produce a valid temporal plan. In other words, exact time values should be assigned to the events. However, these time values cannot be assigned arbitrarily. In fact, there exist certain constraints between these values.

Let the time value assigned to event e_i be $T(e_i)$. Since it is assumed in the generation of a SAT formula, that layer i is ahead of layer $i+1$, a straightforward way to maintain the validity of the final plan is to schedule each event of layer i, before all events of layer $i+1$. Now, suppose that $layer(e_k)=i$, and $layer(e_j)=i+1$ and furthermore, e_j and e_k have no causal relationship with each other. Now, if the order between e_j and e_k is eliminated, not only will the validity of the plan remain unchanged, but also it will be more likely to find a consistent assignment of time values for the events.

In order to pursue the above idea, we define new ordering constrains between different pairs of events. For satisfying all these constraints, we solve an instance of a Simple Temporal Problem (STP) [7]. Each STP is associated with a weighted graph named a Simple Temporal Network (STN). We construct an STN in which each node corresponds to an event. For each i, let x_i be the node that is corresponding to event e_i in the STN. Let ϵ be an arbitrary small rational number and e_j and e_k be different events such that $layer(e_j) \ge layer(e_k)$. The constraint $T(e_j) \ge T(e_k)+\epsilon$ must hold if one of following condition is satisfied:

1. e_k adds a precondition of e_j.
2. e_j deletes a precondition of e_k.

3. either e_j or e_k delete any effect of the other.
4. e_j is a starting event and e_k deletes an all-over condition of *action*(e_j)

Consequently, an edge with the weight $-\epsilon$ will be present in the STN from the node x_k to the node x_j . Moreover, if one of the following conditions holds, the constraint $T(e_j) \geq T(e_k)$ will be necessary, and an edge with weight 0 will be present in the STN from the node x_k to the node x_j.

5. e_k is an ending event and e_j deletes an all-over condition of *action*(e_k).
6. e_j is a starting event and e_k adds an all-over condition of *action*(e_j).

As before, in addition to the constraints stated above, the following constraint must also hold if e_k and e_j are respectively the starting and ending effect of certain action a to ensure that the durations of a in the final plan is set correctly:

7. $T(e_j) \geq T(e_k)+d(a)$
8. $T(e_k)+d(a) \geq T(e_j)$

These constraints will be inserted into the STN by adding an edge with the weight $-d(a)$ from node x_k to node x_j, and another edge with weight $d(a)$ from node x_j to node x_k. We also add a reference node x_0 to the constructed STN which has an edge with weight 0 to every other node. A solution of the STP can be found by computing the length of the shortest path form x_0 to all other nodes. Suppose that such shortest paths exist and the length of the shortest from x_0 to x_i is shown by *distance*(x_0 , x_i). For each event e_i, we assign -*distance*(x_0 , x_i) to $T(x_i)$. Constraints 1 to 8 guarantee that the resulting plan has all the specifications of a valid temporal plan. However, there are situations in which such shortest paths do not exist. It happens when the STN has a negative cycle. In these situations the STP is inconsistent and consequently, all temporal constraints can not be satisfied at the same time. An example of such cases is depicted in figure 2.

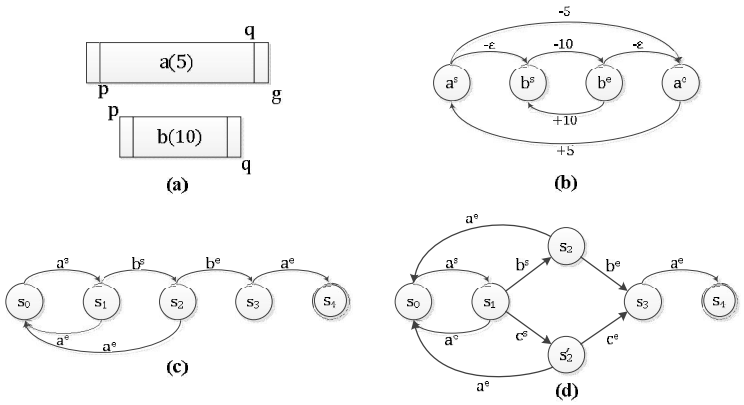

Fig. 2. Negative cycle in an STN, and the detecting FSM

Suppose that 1) action a adds proposition p and g respectively by its starting and ending event. 2) a needs proposition q as a precondition for its ending event. 3) action b that requires p upon beginning and adds q upon ending. 4) durations of actions a, and b, are 5 and 10, respectively. 5) action c is identical to b except for its duration which is 15. 6) The goal of planning is reaching fact g. If a SAT formula with four layers is produced for this problem, then a SAT solver can satisfy the formula by putting the starting event of a in layer 1, the starting event of b in layer 2, the ending event of b in layer 3, and the ending event of a in layer 4. This means action b must be entirely performed inside of action a. However, this situation is impossible considering the fact that duration of a is less than that of b. The abstract plan produced by SAT solver is depicted in figure 2(a). Preconditions and effects of each action are respectively written above and below it.

The invalidity of this plan is caused by the fact that all the durations are abstracted out when the formula is being produced. In fact, the SAT solver assumes that the execution of ending event of a can be postponed, as long as is needed. The STN constructed for the plan of figure 2(a) is depicted in figure 2(b). $a^s b^s b^e a^e a^s$ is a negative cycle with total weight $-5-2\epsilon$. Note that if action b had been replaced by action c in fig 1(a), we would have had another negative cycle with weight $-10-2\epsilon$.

According to definition 7, although planners such as STEP, T-SATPLAN, and TM-LPSAT allow parallel execution of actions, they are in fact using the 1-step encoding. That is because these planners assume simultaneous execution of all events in each step.

Negative Cycle Prevention

As it was stated before, if the STN of an abstract plan has a negative cycle, it cannot be transformed to a valid temporal plan. In such cases, we run the SAT solver again to find a different solution. This can be done by adding an extra blocking clause to the SAT formula so that at least one of the events of invalid plan cannot occur in its current layer. Nevertheless, after adding such a blocking clause, the SAT

solver can produce new plans that are not essentially different from previous one. For instance, consider the example given in figure 2(a). Suppose a^s,b^s,b^e, and a^e are true in layers 1 to 4, respectively. Assume that the SAT formula have 5 layers. If we forbid the exact occurrence of this plan, a new temporally invalid abstract plan can still be produced by shifting a^e to layer 5 and maintaining other events in their current layers. The new solution will have the same negative cycle. In fact, the main cause of the invalidity of the plan has remained unchanged. On the other hand, by replacing action *b* by action *c*, another temporally invalid abstract plan can be produced. This time, the cycle $a^s b^s b^e a^e a^s$ has been replaced by cycle $a^s c^s c^e a^e a^s$. However, since *b* and *c* have identical preconditions and effects, and *c* has a longer duration, we could have inferred from structure of $a^s b^s b^e a^e a^s$ that $a^s c^s c^e a^e a^s$ is also a negative cycle. We now show that negative cycles of particular structure can be prevented more effectively. We also explain how other similar negative cycles can prevented.

The negative cycles in which we are interested are quite similar to the one depicted in figure 2(b). Such cycles happen when several actions of a temporal plan are to be executed consecutively within the execution of another action. These cycles can be regarded as sequences of events. For instance, in the previous example, each time that the a^s, a^e, b^s, and b^e occur with the order $a^s b^s b^e a^e$, we can be sure that the corresponding STN has a negative cycle. Fortunately, such sequences can be detected by using a simple Finite State Machine (FSM). For example, the FSM depicted in figure 2(c) can detect the negative cycle of figure 2(b).

On the other hand, in the STN depicted in figure 2(b) there exists an edge from a^s to b^s only because condition number 1 (stated in section 4.1) is held between the two events. The same condition causes an edge from b^e to a^e. It should be clear that if we replace action *b* by any other action with the same conditions but with a longer duration, the result will be again a negative cycle. Here, action *c* fulfills the stated requirements. We call *c* an alternative for *b*. In fact, ITSAT automatically detects all the alternatives for each action in a given negative cycle. The FSM that detects both $a^s b^s b^e a^e$ and $a^s c^s c^e a^e$ is depicted in figure 2(d). Note that in general, more than one action, each of which having several alternatives, can occur consecutively inside a covering action.

To prevent a cycle, one can start from the first layer and follow the transitions that occur in the FSM. A certain transition occurs in layer *i*, only if the value of the variable corresponding to the label of that transition is equal to one.

If the FSM is constructed, its corresponding negative cycles can be prevented by making the SAT solver simulate the function of the FSM and banning it from being in the terminal state. In order to do so, we must add extra variables and clauses to the SAT formula.

Let $s_0,\ldots,s_m$ be the states of an FSM that detects a certain negative cycle. Suppose that s_0 and s_m are the initial state and the terminal state of the FSM, respectively. Let *T* be the set of all transitions of the FSM, and $label(s_i,s_j)$ denote the event that causes the transition from s_i to s_j. Furthermore, assume that $exit(s_i)$ denotes all the transitions through which the FSM goes from s_i to another state. The following variables and clauses are sufficient for preventing the SAT solver from reproducing negative cycles that are associated with the FSM.

- For every $1 \le i \le n$ and every $0 \le j \le m$, we define a variable $S_{j,i}$. Assigning one to $S_{j,i}$ means that the FSM can be in state *j* after observing all the events that are present in the layers prior to *i*.
- For every $1 \le i \le n$, every $0 \le j \le m$, and every $(s_j,s_k) \in exit(s_j)$, we add a clause $S_{j,i} \wedge e \rightarrow \neg S_{j,i+1} \wedge S_{k,i+1}$, where $e=lable(s_j,s_k)$, to ensure that in each layer, FSM performs only correct transitions.
- For every $1 \le i \le n$ and every $0 \le j \le m$, we add a clause $S_{j,i} \wedge (\neg \bigvee_{e \in E} e) \rightarrow S_{j,i+1}$, where $E=\{label(s_j,s_k) \mid (s_j,s_k) \in exit(s_j)\}$, to ensure that FSM stays in state s_j when it is necessary.
- For every $1 \le i \le n$ and every $0 \le j \le m$, we assert a clause $\neg S_{j,i} \wedge S_{j,i+1} \rightarrow \bigvee_{(k,e) \in N}(S_k \wedge e)$ where $N=\{(k,e) \mid e=lable(s_k,s_j)\}$. This clause prevents the FSM from arbitrarily moving to some state.
- We assert a clause $S_{0,1}$ to ensure that the FSM will start from its initial state in layer one. We also add $\neg S_{j,1}$ for every $1 \le j \le m$, to prevent FSM from being in any other state in layer one.
- For every $1 \le i \le n$, we assert a clause $\neg S_{m,i}$ to ensure that the FSM will never go to its terminal state, detecting a negative cycle.

After constructing the SAT formulation of the FSM, we add it to our previous encoding of the problem. Then, SAT solver is called again to satisfy the new formula. The process of solving the formula and adding negative cycle detectors (i. e., appropriate FSMs) is repeated until a valid plan will be achieved.

Empirical Results

ITSAT has been implemented using C++ programming language. We have used an open source SAT solver, MINISAT2 [9] for solving SAT formulas. To evaluate ITSAT, it has been compared with POPF [5] on the temporal problem sets of recent international planning competitions plus two more domains (i.e., driverlogshift and matchlift) defined by the Strathclyde planning group [10]. POPF is currently one of the most efficient heuristic temporal planners. The experiments were conducted on a 3.1GHz corei5 CPU with 4GB main memory and 30 minutes time-limit for solving each problem. All the results are presented in Table 1.

As it is shown in Table 1, ITSAT significantly outperforms POPF2 in both the total number of solved problems

and the total score. In fact, ITSAT solves 53 more problems than POPF2. Moreover, the total score of ITSAT is 32 percent higher than that of POPF2. These results show a major improvement in temporally expressive temporal planning. The only domains in which POPF2 has a considerable lead over ITSAT are parking and elevators. These two domains are inherently difficult for SAT-based planners, as the number of actions that provide each preposition is relevantly high in their problems.

Domain	IPC	Problems	Solved		Score	
			ITSAT	POPF2	ITSAT	POPF2
zenotravel	2004	20	13	13	**12.41**	11.46
driverlog		20	15	15	**13.70**	11.84
rovers		20	**20**	19	**20**	13.91
depots		22	**13**	7	**12.84**	5.53
airport	2006	50	**37**	15	**37**	13.40
satellite		36	14	14	**13.35**	8.96
pegsol	2011	20	**20**	19	**20**	18.62
crewplanning		20	20	20	19.11	**20**
openstacks		20	8	**20**	6.47	**20**
parking		20	1	**20**	0.53	**20**
elevators		20	0	**2**	0	**2**
floortile		20	**20**	5	**18.78**	5
storage		20	**6**	0	**6**	0
matchcellar		20	20	20	20	20
sokoban		20	**4**	3	**4**	2.89
parcprinter		20	**20**	0	**20**	0
turnandopen		20	6	**9**	6	**8.52**
tms		20	**20**	4	**20**	4
driverlogshift	---	10	10	10	**10**	8.99
matchlift		14	**14**	13	**14**	12.06
total		482	**281**	228	**274.19**	207.18

Table 1. Comparing ITSAT with POPF2

Conclusion

In this report, we introduced a new method for producing SAT encoding of a given temporal planning problem. In the new encoding, durations of all actions are initially abstracted out in order to construct a compact SAT formula. The produced SAT formula is then solved by an off-the-shelf SAT solver. The resulting abstract plan is then relaxed by deleting unnecessary ordering constraints. In order to assign the exact time values to the events of the relaxed plan, based on their order and action durations, a Simple Temporal Network is constructed and resolved. We showed that possible negative cycles of such STNs can be detected by using certain Finite State Machines. We also introduce an automatic method for encoding such FSMs. We showed how adding FSMs encodings to the SAT formula could prevent the SAT solver from reproducing solutions with similar negative cycles. Our empirical results showed that our new planner ITSAT, while not using any planning based heuristic functions, is comparable with POPF, which is currently the state-of-the-art in non-optimal temporal planning.

References

[1] Huang, R., Chen, Y., and Zhang, W., An optimal temporally expressive planner: Initial results and application to P2P network optimization, *Proceedings of 19th International Conference on Automated Planning and Scheduling*, AAAI press, 2009.

[2] Mali, A.D., and Liu, Y., T-SATPLAN: A SAT-based Temporal Planner, *International Journal of Artificial Intelligence Tools* 15(5): 779-802, 2006.

[3] Cushing, W., Kambhampati, and Weld, D.S., When is temporal planning really temporal?, *Proceedings of 20th International Joint Conference on Artificial Intelligence*, 1852-1859, AAAI press, 2007.

[4] Halsey, K., Long, D., and Fox, M. , Managing Concurrency in Planning Using Planner-Scheduler interaction, *Artificial Intelligence* 173(1): 1-44, 2009.

[5] Coles, A.J., Coles, A., Fox, M., and Long, D., Forward-Chaining Partial-Order Planning, *Proceedings of 20th International Conference on Automated Planning and Scheduling*, 42-49, AAAI press, 2010.

[6] Hoffmann, J., Nebel, B., The FF Planning System: Fast Plan Generation Through Heuristic Search, *Journal of Artificial Intelligence Research* 14: 253-302, 2001.

[7] Dechter, R., Meiri, I., and Pearl, J., Temporal Constraint Networks, *Artificial Intelligence* 49(1-3): 61-95, 1991.

[8] Fox, M. and Long, D., PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains, *Journal of Artificial Intelligence Research* 20: 61-124, 2003.

[9] Eén, N., and Sörensson, N., An extensible SAT-solver, *Proceedings of 6th International Conference on Theory and Applications of Satisfiability Testing*, 502-518, Springer, 2003.

[10] Strathclyde Planning Group, http://planning.cis.strath.ac.uk