

Encoding Temporal Planning as CSP

Amol D. Mali

Electrical Engg. & Computer Science

University of Wisconsin, Milwaukee, WI 53211

Phone: 1-414-229-6762, Fax: 1-414-229-2769, mali@miller.cs.uwm.edu

Abstract

(Appears in the proceedings of the AIPS workshop on planning in temporal domains, Toulouse, France, April 2002, pp. 18-25.)

Recent advances in constraint satisfaction and heuristic search have made it possible to solve classical planning problems significantly faster. There is an increasing amount of work on extending these advances to solving more expressive planning problems which contain metric time, quantifiers and resource quantities. One can broadly classify classical planners into two categories: (i) planners doing refinement search and (ii) planners iteratively processing a representation of finite size like a SAT encoding or planning graph or a constraint satisfaction problem (CSP). One key challenge in the development of planners casting planning as SAT or CSP is the identification of constraints which are satisfied if and only if there is a plan of k steps. This task is even more complex for planners handling metric time and/or resource quantities and/or quantifiers. In this paper we show how such a SAT encoding can be synthesized for temporal planning. This encoding contains twenty kinds of constraints. We show how this encoding can be simplified. Though solving a SAT encoding may not be the best approach to temporal planning (especially when there are too many actions and/or too much variation in durations of actions), the set of constraints we identify makes it easier to develop temporal planners casting planning as a constraint satisfaction problem other than SAT, like integer linear programming (ILP). The SAT encoding we present can be easily adapted to more complex cases of temporal planning like the one in which different pre-conditions and effects of an action may be true at different times during its execution.

1 Introduction

Recent advances in classical planning have made it possible to solve larger planning problems than before. Many of the recently developed planners cast planning as a constraint satisfaction problem. The planner in [Do & Kambhampati 2000] solves a planning problem by solving a CSP generated based on the planning

graph built by Graphplan [Blum & Furst 1997]. The CPlan planner [van Beek & Chen 1999] solves planning as a constraint programming problem. Graphplan [Blum & Furst 1997] builds a representation called planning graph and identifies mutual exclusion constraints (mutexes) between actions. Graphplan propagates these mutexes to identify more mutexes. These inferred mutexes significantly improve the backward search over planning graph for solution extraction. SAT is a specific kind of CSP in which all variables are boolean and the constraints involve variables connected with operators from boolean logic. Some of the recently developed efficient planners cast planning as propositional satisfiability. These include SAT-plan [Kautz & Selman 1996], [Kautz et al 1996], MEDIC [Ernst et al 1997] and the planner which casts hierarchical task network planning as satisfiability [Mali 1999],[Mali 2000]. SAT encodings of a large number of planning problems in benchmark domains contain a significant number of binary clauses. Simplification techniques for binary clauses have been shown to improve the performance of planning as SAT [Brafman 2001]. Advances in SAT solving like better branching heuristics [Li & Anbulagan 1997] can be exploited to further improve the performance of SAT-based planners. Some of the recently developed efficient planners cast planning as 0-1 integer linear programming (ILP) in which all variables are boolean and all constraints are linear. These planners include the planner from [Vossen et al 2000].

More expressive planning problems contain quantifiers, conditional effects, metric time and resource quantities. Examples of such problems include many NASA planning applications [Smith & Weld 1999]. In these applications, both spacecraft and planetary rovers use heaters to warm up various components and these actions may span several other actions or experiments [Smith & Weld 1999]. There is an increasing interest in extending/adapting advances in classical planning to solving more expressive planning problems.

Temporal Graphplan (TGP) is an extension of Graphplan [Blum & Furst 1997] to handle time durations of actions. The LPSAT planner [Wolfman & Weld 1999] solves planning problems involving resource quantities by transforming them into problems containing linear constraints and propositional clauses. Some of the constraints solved by LPSAT have a logical and a mathematical part, e.g. the constraint $((a > 3) \Rightarrow (x \vee y))$. Other more expressive planners include [Tsamardinos et al 2000].

Development of more expressive planners involves several challenges. These include verification of soundness and completeness, besides getting optimal plans in a shorter time. The tasks of ensuring soundness and completeness are trivial for refinement planners. This is because refinement planners maintain a different representation for different partial plans in the form of nodes in a search tree. In progression, an action sequence is executed starting at initial state and it is checked whether goal is true at the end of its execution. A partial plan is a set of constraints which may or may not be a plan. Partial plans generated by forward state-space planners and backward state-space planners are sequences of actions whose goal achieving capability can be verified by simple methods like progression. A partial plan generated by partial order planner contains a set of constraints like step-action bindings and partial orderings over steps. The goal achieving capability of such a partial plan can be verified simply by finding if there exists a total order over the steps which is consistent with the partial order constraints in the partial plan such that the goal is achieved when the steps are executed in the order specified by the total order. The verification of soundness and completeness of more expressive planners which cast planning as CSP is non-trivial since a set of constraints needs to be identified such that these are solved if and only if there is a plan of k steps. This set needs to be loose enough to allow generation of all action sequences that are plans and tight enough to exclude generation of any action sequence that is not a plan. The correctness of the temporal planner TGP [Smith & Weld 1999] is difficult to verify.

The adaptation of SAT-plan to handle metric time is challenging because of task of identifying all constraints that must be satisfied if and only if there is a plan of bounded length. Temporal planning problem is the problem of finding a set of $\langle action_i, start_time_i \rangle$ tuples for achieving a given goal, starting with a given completely described initial state using actions that have time durations. $start_time_i$ is the start time of action $action_i$. This is the time at which the execution of $action_i$ starts.

We use the following assumptions in temporal planning from [Smith & Weld 1999] when setting up the SAT encoding. Effects of actions are undefined during their execution. Action durations are integers. Pre-conditions of an action in a plan should all be true at its start time. The effects of an action hold only at the end of its execution. The SAT encoding is such that it has a solution if and only if there is a plan of k time steps. In particular, we identify twenty kinds of constraints which together form the encoding when translated into clauses. We also show how the encoding can be simplified so that the number of clauses and variables are reduced without losing soundness and completeness. We show how the encoding can be adapted to more complex case of temporal planning in which it is not necessary for different pre-conditions of an action to be true at same time and it is not necessary for different effects to hold at the same time. We show how temporal planning can be cast as a CSP other than SAT.

2 Background

In this section, we explain how SAT-based planners work and describe the state-space SAT encoding from [Kautz et al 1996] for classical planning where all actions have unit duration. An action is a ground instance of an operator. For example, if $move(x, y, z)$ is an operator for moving block x from top of block y or table to top of block z or table, $x \neq y, y \neq z, x \neq z, x \neq Table$, then there are $O(n^3)$ actions when there are n blocks.

SAT-plan [Kautz & Selman 1996] works in the following manner. Based on initial state, number of steps assumed to exist in plan (k), goal state and action description, it generates an encoding (SAT instance) such that the instance has a model if and only if there is a plan of k steps. The steps range from 0 to $(k - 1)$. The encoding is simplified by rules of inference of boolean logic, e.g. $a \wedge (\neg a \vee b)$ can be simplified to $(a \wedge b)$ (this simplification step is optional but most SAT planners carry this out). The encoding is then passed to a SAT solver. If it is solved, the solution (truth assignment) is interpreted and plan is output by reading the truth values assigned to step-action binding and step ordering variables. If it cannot be solved, then value of k is increased and the process of encoding generation, simplification and solving is repeated. Variables representing an occurrence of actions at various steps are step-action binding variables.

The explanatory frame axiom-based state-space encoding [Kautz et al 1996] contains the following constraints: (i) All propositions true in the initial state are true at time 0 and all propositions false in the ini-

tial state are false at time 0. (ii) If an action occurs at time t , its pre-conditions are true at time t and its effects are true at time $(t+1)$. (iii) If an action o_i needs proposition p true and action o_j deletes p , then o_i and o_j cannot occur at same time. (iv) All propositions from goal are true at time k . (v) If a proposition p is true at time t and false at time $(t+1)$, some action deleting p must occur at time t . If a proposition p is false at time t and true at time $(t+1)$, some action making p true must occur at time t . These constraints are included to ensure that truth of a proposition cannot change unless an action causing the change occurs. These constraints are known as explanatory frame axioms.

3 SAT Encoding for Temporal Planning

To generate a SAT encoding, we first need to bound the number of steps in plan at which actions occur. We denote this bound by k . O denotes the set of all actions in domain and U denotes the set of all ground fluents in domain. A fluent is a proposition whose truth can change. Propositions that are not made true or false by any action are considered to be invariants whose truth remains unchanged. Before explaining the constraints appear in the propositional encoding of a temporal planning problem, we explain some relevant notation for various boolean variables in the encoding. $pt(t)$ is a boolean variable which denotes the truth of true state of fluent p at time t . If $pt(t)$ is assigned true, it means that fluent p is true at time t . If $pt(t)$ is assigned false, it means that the fluent p is not true at time t (so it is either false or undefined/unknown). $pu(t)$ denotes that the truth of the undefined state of fluent p at time t . If $pu(t)$ is assigned true, it means that the fluent p is undefined at time t . If $pu(t)$ is assigned false, it means that the fluent p is not undefined at time t . In this case p is either true or false at time t . $pf(t)$ denotes that the truth of the false state of fluent p at time t . If $pf(t)$ is assigned true, it means that the fluent p is false at time t . If $pf(t)$ is assigned false, it means that the fluent p is not false at time t . In this case p is either true or undefined at time t . $o_i(t)$ denotes the occurrence of the ground action o_i at time t . $o_i(t)$ is an action variable. $pf(t)$, $pt(t)$ and $pu(t)$ are fluent variables. If $o_i(t)$ is assigned true, it means that o_i occurs at time t . If $o_i(t)$ is assigned false, it means that o_i does not occur at time t . d_i denotes the duration of the ground action o_i . The time steps in the encoding at which actions may occur range from 0 to $(k-1)$. d_{max} is the maximum of durations of all actions. The following constraints appear in the state space encoding of a temporal planning problem. The encoding is a state-space encoding since it refers

to states of all fluents at all time steps. We assume that actions do not have negated pre-conditions.

1. The constraints of this kind state that each fluent is either false or true or undefined at every time step. For all fluents p , for all times $t \in [0, k]$, $(pt(t) \vee pf(t) \vee pu(t))$. There are $(k+1) \cdot |U|$ clauses of this kind. These contain $3 \cdot (k+1) \cdot |U|$ variables.
2. The constraints of this kind state that at a time, a fluent cannot be in more than one of the following states: true, false and undefined. For all fluents p , for all times $t \in [0, k]$, $(pt(t) \Rightarrow \neg pf(t))$, $(pt(t) \Rightarrow \neg pu(t))$, $(pf(t) \Rightarrow \neg pu(t))$. There are $3 \cdot (k+1) \cdot |U|$ clauses of this kind.
3. Fluents true in initial state are true at time 0. Fluents false in initial state are false at time 0. These constraints contribute $|U|$ unit clauses to the encoding. If the initial state is $(a \wedge b \wedge \neg c \wedge \neg d)$, the encoding contains $(at(0) \wedge bt(0) \wedge cf(0) \wedge df(0))$.
4. Fluents from goal are true at time k . If goal is a conjunction of s fluents, these constraints contribute s unit clauses to the encoding.
5. If an action o_i occurs (starts) at time t , $0 \leq t \leq (k - d_i)$, its pre-conditions are true at t and its effects are true at time $(t + d_i)$. These constraints contribute $O(k \cdot |O| \cdot m)$ clauses to the encoding, m being the maximum sum of the number of pre-conditions and the number of effects of an action. These constraints contribute $k \cdot |O|$ variables to the encoding. For example, if x and y are pre-conditions of action o_3 , and $\neg x$ and z are its effects, this constraint generates $(o_3(j) \Rightarrow (xt(j) \wedge yt(j) \wedge xf(j+d_3) \wedge zt(j+d_3)))$, where $j \in [0, k - d_3]$. Note that we require $t \leq (k - d_i)$ because if o_i starts at $t > (k - d_i)$, it will finish at time $(k+1)$ or later and the encoding has only $(k+1)$ time steps ranging from 0 to k .
6. If an action o_i does not delete any of its pre-conditions, then if o_i starts at time t , where $0 \leq t \leq (k - d_i)$, then o_i cannot start at any time $t' \in [t+1, (t + d_i - 1)]$, d_i being duration of o_i . These constraints prevent an occurrence of an action during its execution, when the action does not delete any of its pre-conditions. These can be considered as mutual exclusion relationships of actions with themselves. These constraints contribute $O(k \cdot q \cdot d')$ clauses to the encoding, where q is the number of actions which do not delete any of their own pre-conditions and d' is the maximum of the durations of these actions.
7. If an action o_i does not delete its pre-condition x , then if o_i starts at time t , x remains true over the closed interval $[t+1, t+d_i]$ when $d_i > 1$. If there are m actions that do not delete any of their own pre-conditions, these constraints contribute $O(m \cdot d_{max} \cdot k \cdot m')$ clauses

to the encoding, where m' is the maximum number of pre-conditions of an action not deleted by the action.

8. The pre-conditions of an action o_i which are deleted by o_i are undefined over the closed interval $[t + 1, t + d_i - 1]$ if o_i starts at t . If there are m_1 actions that delete some of their own pre-conditions, these constraints contribute $O(m_1 \cdot d_{max} \cdot k \cdot m'')$ clauses to the encoding, where m'' is the maximum number of pre-conditions of an action deleted by the action. For example, if effects of action o_4 are $\neg p$ and q , such that p is one of its pre-conditions, and duration of o_4 is 3, then this constraint yields $(o_4(j) \Rightarrow (pu(j+1) \wedge pu(j+2)))$.

9. Effects of an action o_i (except pre-conditions of o_i which o_i deletes) are undefined from time $(t + 1)$ until time $(t + d_i - 1)$, including both these times, if o_i starts at t . These constraints contribute $O(|O| \cdot d_{max} \cdot k \cdot m''')$ clauses to the encoding, where m''' is the maximum number of effects of an action, excluding the pre-conditions deleted by it. For example, if effects of action o_4 are $\neg p$ and q , such that p is one of its pre-conditions, and duration of o_4 is 3, then this constraint yields $(o_4(j) \Rightarrow (qu(j+1) \wedge qu(j+2)))$.

Constraints 10, 11, 12, 13, 14 and 15 are all explanatory frame axioms. These contribute $O(k \cdot |U|)$ clauses to the encoding. These constraints prevent changes in states of fluents without occurrences of actions causing these changes.

10. If fluent p is true at time t and false at time $(t+1)$, some action of duration 1 which deletes p must occur (start) at time t . For example, if actions o_4, o_6 and o_9 are the only actions of duration 1 which delete fluent x , this constraint generates $((xt(t) \wedge xf(t+1)) \Rightarrow (o_4(t) \vee o_6(t) \vee o_9(t)))$.

11. If fluent p is false at time t and true at time $(t+1)$, some action of duration 1 which makes p true must occur (start) at time t .

12. If fluent p is true at time t and undefined at time $(t + 1)$, some action of duration greater than 1 which deletes p must occur (start) at time t . For example, if $k = 20$ and the only actions of duration more than 1 which delete fluent x are o_7, o_8 and o_{11} such that $d_7 = 4, d_8 = 12$ and $d_{11} = 7$, then we have $((xt(t) \wedge xu(t+1)) \Rightarrow (o_7(t) \vee o_8(t) \vee o_{11}(t))), t \in [0, 8]$. We have $((xt(t) \wedge xu(t+1)) \Rightarrow (o_7(t) \vee o_{11}(t))), t \in [9, 13]$. We have $((xt(t) \wedge xu(t+1)) \Rightarrow o_7(t)), t \in [14, 16]$. This constraint does not lead to any clauses for the change in state of x from true to unknown for $t \in [17, 19]$. This is because any action changing x in this fashion will end after $t = 20$ in case it starts at time 17 or later.

13. If fluent p is false at time $t \in [0, k - 2]$ and undefined at time $(t + 1)$, some action of duration greater than 1 which makes p true must occur (start) at time

t .

14. If fluent p is undefined at time $t \in [0, k - 1]$ and false at time $(t + 1)$, some action o_i of duration greater than 1 which makes p false must start at time $(t + 1 - d_i)$. For example, if o_5, o_7 and o_{10} are the only actions which delete fluent y and $d_5 = 2, d_7 = 5$ and $d_{10} = 7$, then this constraint is represented by $((yu(t) \wedge yf(t + 1)) \Rightarrow (o_5(t - 1) \vee o_7(t - 4) \vee o_{10}(t - 6)))$.

15. If fluent p is undefined at time $t \in [0, k - 1]$ and true at time $(t + 1)$, some action o_i with duration greater than 1 which makes p true must start at time $(t + 1 - d_i)$.

All of the following constraints represent mutual exclusion relations between actions. These contribute $O(k \cdot |O|^2)$ clauses to the encoding. These constraints are identified after taking into account the all possible kinds of temporal relationships between two actions (seven types) from Figure 1 and all possible effects two actions may have on a fluent p . Given two actions o_i and o_j and a fluent p , the actions can affect the fluent in the following ways: (i) both o_i and o_j need p and both delete p , (ii) only o_i needs p and both o_i and o_j delete p , (iii) neither o_i nor o_j needs p and both delete p , (iv) neither o_i nor o_j needs p and both make p true, (v) o_i makes p true and o_j only deletes p , (vi) o_i only needs p and o_j only deletes p and, (vii) o_i makes p true and o_j both needs p and deletes p .

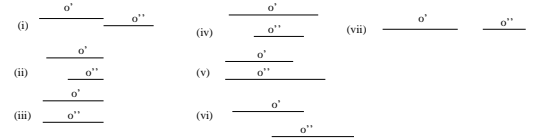


Figure 1: Temporal relations between two actions o' and o''

16. If two actions o_i and o_j have same duration such that o_i deletes pre-condition of o_j , then o_i and o_j cannot start at same time t . The clauses $\neg(o_i(t) \wedge o_j(t)), t \in [0, k - 1]$ are generated for all pairs of such actions to represent this constraint in the encoding.

17. If o_i with duration greater than 1 deletes pre-condition p of action o_j which has duration 1 such that o_j does not delete its own pre-condition p , then o_i and o_j cannot start at same time t . The clauses $\neg(o_i(t) \wedge o_j(t)), t \in [0, k - 1]$ are generated for all pairs of such actions to represent this constraint in the encoding.

18. If o_i has pre-condition p which it also deletes and action o_j only deletes p (o_j does not need p), then if o_i starts at t , then o_j cannot start at a time $t' \geq t$ if $(t' + d_j) = (t + d_i)$. To represent this in the encoding, the clauses $\neg(o_i(t) \wedge o_j(t'))$ are generated for all pairs of such actions, for all times t and t' such that $(t + d_i) = (t' + d_j), t' \geq t, t \in [0, k - 1], t' \in [0, k - 1]$.

19. If actions o_i and o_j both only delete some fluent p and both do not need p , then it cannot be the case that o_i ends at starting time of o_j . This avoids consecutive deletions of a fluent without making the fluent true. To represent this in the encoding, the clauses $\neg(o_i(t) \wedge o_j(t'))$ are generated for all pairs of such actions, for all times t and t' such that $(t + d_i) = t', t \in [0, k - 1], t' \in [0, k - 1]$.

20. If two actions o_i and o_j both only delete some fluent p and both do not need it, then it cannot be the case that o_i ends at ending time of o_j . To represent this in the encoding, the clauses $\neg(o_i(t) \wedge o_j(t'))$ are generated for all pairs of such actions, for all times t and t' such that $(t + d_i) = (t' + d_j), t \in [0, k - 1], t' \in [0, k - 1]$.

The encoding contains $O(k \cdot (|O| + |U|))$ variables and $O(k \cdot (|O| + |U|) + k \cdot |O|^2)$ clauses. The number of variables and clauses in the SAT encoding of a temporal planning problem are respectively higher than the the number of variables and clauses in the encoding of the same problem with all action durations set to 1. This is mainly because of the action durations and the three variables needed to model three different states of each fluent.

4 Discussion

We showed how temporal planning can be encoded as a SAT problem and found the asymptotic number of variables and clauses in the encoding. In this section, we show how the size of this encoding can be reduced without losing soundness and completeness. We also show how this encoding can be adapted to the more complex case of temporal planning in which it is not necessary for all pre-conditions of an action to be true at the same time and different effects of an action may become true at different times during its execution. We also show how the encoding is useful in casting temporal planning as a CSP other than SAT.

4.1 Reducing Encoding Size

Though a smaller encoding is not always easier to solve, smaller encodings have been shown to be solvable faster [Kautz & Selman 1996], [Ernst et al 1997], [Mali 1999]. The size of an encoding can be reduced by propagating the truth assigned to unit clauses. Such unit clauses are available in the specification of initial state and goal in the encoding. Almost all SAT simplification techniques which apply to the case where all actions have duration 1 also apply to the case where actions have unequal durations and we will not discuss these.

One can select a better value of k to avoid solving several encodings generated with unacceptable lower

values of k . The planning graph [Blum & Furst 1997] can be used to wisely choose the value of k . Graphplan works in 2 phases. The first involves growing a **planning graph** and is called the **plangraph** construction phase. This is a forward phase, beginning with the initial state. The second phase is a solution extraction phase. This is backward search phase starting with the goal. Plangraph, or planning graph (PG), has two kinds of levels called **action levels** and **proposition levels**. The 0th proposition level is the same as the initial state. The 0th proposition level occurs before the 0th action level which in turn occurs before the 1st proposition level which precedes the 1st action level, etc. In general, the i th proposition level is immediately succeeded by the i th action level. And, the i th action level immediately precedes $(i + 1)$ th proposition level. A proposition level and an action level can be considered as sets whose members are the same as the contents of these levels. The i th action level in the plangraph contains all actions whose all pre-conditions appear in the i th proposition level. There is also a dummy action called a **no-op**, **maintenance action**, or **persistence action** in the i th action level for each proposition in the i th proposition level. The pre-condition and effect of this action is the proposition for which the action was created. This action is included in the plangraph because if no action changing the truth of the proposition occurs, the truth of the proposition remains same. The $(i + 1)$ th proposition level is the union of the i th proposition level and the effects of the actions in the i th action level. Thus, proposition level i is a superset of proposition level $(i - 1)$. Similarly, action level i is a superset of action level $(i - 1)$.

There are three kinds of edges in plangraph: **(i)** edges from propositions in proposition level i to the same propositions in proposition level $(i + 1)$ (for no-ops), **(ii)** edges from propositions in proposition level i to actions (whose pre-condition list contains these propositions) in action level i and **(iii)** edges from actions in action level i to propositions (which are effects of these actions) in proposition level $(i + 1)$.

A key to the efficiency of Graphplan is the inference of binary **mutex** (mutually exclusive) relations. Two kinds of mutexes are found: (i) mutexes between actions, and (ii) mutexes between propositions. Each of these two kinds of mutexes could be static or dynamic. Static mutexes are found by examining pre-conditions and effects of actions. Dynamic mutexes are found by propagating static mutexes using truths of conditions in the initial state. Note that dynamic mutexes may be permanent or temporary. Two actions at action level i are mutex if (i) their effects are inconsistent (the effect

of one action is the negation of some effect of another action), or (ii) one action deletes some pre-condition of another action, or (iii) the actions have pre-conditions that are mutex at proposition level i . For (iii) one needs to understand the definition of mutex propositions given next. Two propositions p, q in proposition level i are mutex if (i) p is the negation of q , or (ii) all ways (actions) of achieving p are mutex with all ways (actions) of achieving q . Note that while considering all ways of achieving a proposition, no-ops are also considered.

If the plangraph has a proposition level that contains all conditions from the goal such that no two of these are mutex, Graphplan starts a backward search for a plan. For each condition in the goal, it chooses a source of support (an action) in the immediately preceding action level. The pre-conditions of these actions become subgoals to be achieved. If no two pre-conditions of the chosen actions are mutex, it chooses sources of support for these subgoals from the immediately preceding action level and continues this process. In case subgoals are found to be mutex, it backtracks, chooses different sources of support, and repeats this process. If all combinations of the supporting actions fail for each subgoal at each proposition level, then Graphplan grows plangraph with one more action level and proposition level and tries the backward solution extraction process again. If no solution is found, Graphplan extends planning graph by one action level and 1 proposition level and tries the solution extraction again. Graphplan is guaranteed to report unsolvability of a problem. A plangraph is said to **level off** if the none of the following change when the plangraph is grown further: (i) the number of actions in last action level, (ii) the number of propositions in last proposition level, (iii) the number of pairs of actions that are mutex in last action level, and (iv) the number of pairs of mutex propositions in last proposition level. In the planning graph in Fig. 2, false propositions are not shown in proposition levels for readability. Because of same reason, mutex relations are not shown. The planning graph has 3 proposition levels and 2 action levels. M1, M2, M3 and M4 are actions in the domain. The pre-conditions and effects of these actions are shown on the left and right sides of the boxes respectively. Here is the planning graph figure. The goal is achievable with the plan Step 0: M2, Step 1: M1 & M3. The actions M1 and M2 are static mutex and the actions M3, M4 are dynamic mutex in the first action level. Kautz & Selman have shown that planning graph can be used to reduce the size of a SAT encoding for planning with actions of unit duration [Kautz & Selman 1999].

It is clear that actions that do not appear in the

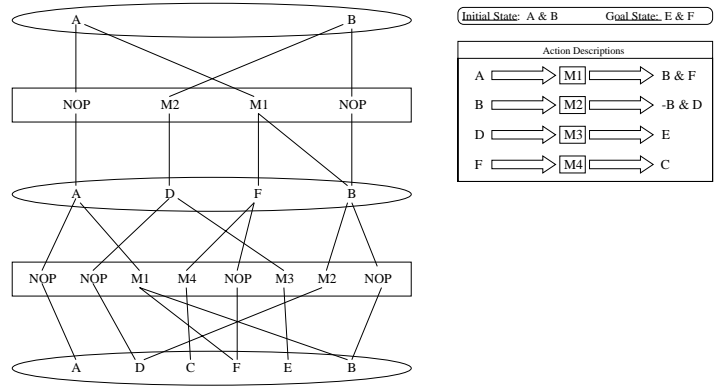


Figure 2: A planning graph

last action level of the leveled off planning graph are not relevant to solving the problem. Such actions need not be represented in the SAT encoding. Let us consider a leveled off planning graph with four action levels which are $\{o_2, o_4\}$, $\{o_2, o_4, o_5\}$, $\{o_2, o_4, o_5, o_8, o_9\}$, and, $\{o_2, o_4, o_5, o_8, o_9, o_{11}\}$ respectively, excluding no-ops from the action levels. This planning graph is constructed by Graphplan [Blum & Furst 1997]. If $O = \{o_i \mid 1 \leq i \leq 20\}$, then it can be concluded that actions $o_1, o_3, o_6, o_7, o_{10}, o_{12}, o_{13}, o_{14}, o_{15}, o_{16}, o_{17}, o_{18}, o_{19}$ and o_{20} are not relevant to solving the problem. These need not be represented in the encoding for temporal planning. Since o_5 occurs in the first action level of the planning graph, it is clear that in case o_5 occurs in temporal plan, its start time will be greater than or equal to $\min(d_2, d_4)$. This is because the planning graph shows that o_5 can occur only after one or more of the actions from $\{o_2, o_4\}$ occur. This means that we need not create variables $o_5(t), t \in [0, \min(d_2, d_4) - 1]$. Similar argument shows that we need not create the following variables: $o_8(t_1), t_1 \in [0, (\min(d_2, d_4) + \min(d_2, d_4, d_5)) - 1]$, $o_9(t_2), t_2 \in [0, (\min(d_2, d_4) + \min(d_2, d_4, d_5)) - 1]$ and, $o_{11}(t_3), t_3 \in [0, (\min(d_2, d_4) + \min(d_2, d_4, d_5) + \min(d_2, d_4, d_5, d_8, d_9)) - 1]$. This significantly reduces the number of variables and clauses in the encoding. This also reduces the number of literals in various clauses. The number of steps k can then be chosen so that $\theta' \leq k \leq \theta''$ where $\theta' = (\min(d_2, d_4) + \min(d_2, d_4, d_5) + \min(d_2, d_4, d_5, d_8, d_9) + \min(d_2, d_4, d_5, d_8, d_9, d_{11}))$ and $\theta'' = (\max(d_2, d_4) + \max(d_2, d_4, d_5) + \max(d_2, d_4, d_5, d_8, d_9) + \max(d_2, d_4, d_5, d_8, d_9, d_{11}))$. Note that k is not bounded from below by θ' . k is not bounded from above by θ'' . Higher value of k can be chosen if an encoding with $k = \theta''$ is found to be unsolvable.

4.2 More Complex Temporal Planning

We assumed that all pre-conditions of an action are true at the same time which is same as the time of start of execution of the action. We also assumed that all effects of an action hold at the same time which is same as the time of end of execution of the action. In reality, different effects may hold at different times and it may not be necessary for all pre-conditions of an action to hold at the same time. Consider the action $move(A, B, C)$ which moves block A from top of block B to top of block C . Its pre-conditions are $clear(A), on(A, B)$ and $clear(C)$. Let us assume that there are several grippers and one does not need the pre-condition $hand - empty$. The effects of this action are $on(A, C), \neg clear(C), clear(B)$ and, $\neg on(A, B)$. $clear(A)$ and $on(A, B)$ have to be true at same time. However $clear(C)$ can become true later during the execution of the action since some other action may move block from the top of C before A is put on C . The effect $\neg on(A, B)$ holds immediately and the effect $on(A, C)$ becomes true much later. Consider the action $fly(P, London, Paris)$ which flies plane P from $London$ to $Paris$. Its effects are $at(P, Paris), \neg at(P, London)$. It is clear that $\neg at(P, London)$ becomes true much earlier than $at(P, Paris)$. Such an action may be specified with times at which various pre-conditions are needed true relative to s and various effects become true, relative to s , where s is start time of the action.

Let us see whether one needs to change constraints from section 3 to handle such actions and if so how. Constraints 1 through 4 do not need any change to handle such actions. Constraint 5 needs a minor change. The new constraint specifies that different pre-conditions and effects are true at different times. Constraints 6 and 7 do not need any change. Constraint 8 needs a minor change. The new constraint states that pre-condition of an action deleted by itself is undefined over the interval $[t+1, t+r-1]$, when the pre-condition is deleted r time units after the start of the action, t being start time of the action. Constraint 9 needs a minor change to specify that different effects of an action are undefined over different time intervals. Constraints 10 and 11 do not need any change. Constraint 12 needs a minor change. The new constraint states that if a fluent p is true at time t and undefined at time $(t+1)$, some action of duration greater than 1 which deletes p at two time units or more later than the time of start of its execution must occur at time t . The remaining constraints can be easily adapted to handle such actions.

4.3 Temporal Planning as CSP other than SAT

Since SAT can be easily transformed into 0-1 ILP, an ILP encoding of temporal planning can be directly created using our SAT encoding. For example, the clause $(x \vee y \vee \neg z \vee \neg b)$ can be translated into the linear constraint $(x' + y' + (1 - z') + (1 - b')) \geq 1$ where the domain of the integer variables x', y', z', b' is $[0, 1]$. The direct translation of our SAT encoding into 0-1 ILP encoding leads to $O(k \cdot (|O| + |U|))$ integer variables and $O(k \cdot (|O| + |U|) + k \cdot |O|^2)$ linear constraints among these. It has been empirically shown in [Vossen et al 2000] that the ILP formulations need to be stronger to be easily solvable. The strength of an ILP formulation obtained by a direct translation of SAT can be improved by adding constraints so as to reduce the number of non-integer solutions of the ILP formulation.

One can create a CSP using the constraints that lead to the SAT encoding. Specifically, one can have $k \cdot |O|$ boolean variables whose values represent occurrence/absence of actions at various time steps. One can have $(k+1) \cdot |U|$ variables with the domain $\{t, f, u\}$ to represent various states of the fluents at all time steps. Note that in SAT encoding we need $3 \cdot (k+1) \cdot |U|$ boolean variables to represent various states of all fluents at all time steps. In the CSP formulation, we need only $(k+1) \cdot |U|$ variables. Though the domains of these variables contain 3 values, the worst-case size of the space of assignments to the fluent variables is lower in the CSP formulation. Since by definition each variable in CSP is assigned a unique value, we do not need constraints to specify that a fluent cannot be in more than one state at any time. Note that in the SAT encoding we need $3 \cdot (k+1) \cdot |U|$ binary clauses to specify that a fluent cannot be in more than one state at any time. In SAT encoding, we need $(k+1) \cdot |U|$ clauses to specify that each fluent is true or false or undefined at each time step. No constraints are needed in the CSP to specify this requirement. Remaining constraints leading to the SAT encoding can be translated to complete the CSP formulation. This shows how the constraints we developed to generate a SAT encoding are useful in casting temporal planning as a CSP. Note that no extra effort is needed to verify the correctness of the CSP formulation.

5 Summary

There is an increasing amount of work on extending/adapting the recent advances in plan synthesis under classical assumptions to more expressive planning. More expressive planning involves dealing with metric time, conditional effects, quantifiers and resource

quantities. Verifying the soundness and completeness of such planners which cast planning as some kind of CSP is non-trivial. We developed a SAT encoding for temporal planning such that the encoding is solvable if and only if there is a plan whose execution time is less than or equal to chosen bound ($k + 1$). We showed how the size of the encoding can be significantly reduced using information in planning graph. We showed how the SAT encoding can be adapted to handle actions all of whose pre-conditions need not be true at same time and/or whose effects become true at different times. We also showed how the SAT encoding is useful in casting temporal planning as a 0-1 ILP and as a CSP different from SAT and 0-1 ILP.

Acknowledgement - This work is supported in part by NSF grant IIS-0119630 to Mali. Mali also thanks students in his graduate course "Artificial Intelligence Planning Techniques" (CS-790-001) taught in Fall 2000 and Fall 2001 for useful discussions.

References

- [**van Beek & Chen 1999**] Peter van Beek and Xinguang Chen, CPlan: A constraint programming approach to planning, Proceedings of National Conference on Artificial Intelligence (AAAI), 1999.
- [**Blum & Furst 1997**] Avrim Blum and Merrick Furst, Fast planning through planning graph analysis, Artificial Intelligence 90, 1997, 281-300.
- [**Brafman 2001**] Ronen I. Brafman, A simplifier for propositional formulas with many binary clauses, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2001.
- [**Do & Kambhampati 2000**] Minh B. Do and Subbarao Kambhampati, Solving planning graph by compiling it into CSP, Proceedings of international conference on artificial intelligence planning and scheduling (AIPS), 2000.
- [**Ernst et al 1997**] Michael Ernst, Todd Millstein and Daniel Weld, Automatic SAT compilation of planning problems, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997.
- [**Kautz & Selman 1996**] Henry Kautz and Bart Selman, Pushing the envelope: Planning, propositional logic and stochastic search, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1996.
- [**Kautz et al 1996**] Henry Kautz, David McAllester and Bart Selman, Encoding plans in propositional logic, Proceedings of Knowledge Representation and Reasoning conference (KR), 1996.
- [**Kautz & Selman 1999**] Henry Kautz and Bart Selman, Unifying SAT-based and graph-based planning, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1999.
- [**Li & Anbulagan 1997**] Chu Min Li and Anbulagan, Heuristics based on unit propagation for satisfiability problems, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997.
- [**Mali 1999**] Amol Mali, Plan merging and plan reuse as satisfiability, Proceedings of European Conference on Planning (ECP), Durham, UK, 1999.
- [**Mali 2000**] Amol Mali, Enhancing HTN planning as satisfiability, Proceedings of the International Conference on Artificial Intelligence and Soft Computing (ASC), Banff, Alberta, Canada, 2000.
- [**Mali & Kambhampati 1998**] Amol Mali and Subbarao Kambhampati, Encoding HTN planning in propositional logic, Proceedings of the international conference on Artificial Intelligence Planning Systems (AIPS), 1998.
- [**Smith & Weld 1999**] David E. Smith and Daniel S. Weld, Temporal planning with mutual exclusion reasoning, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1999.
- [**Tsamardinos et al 2000**] Ioannis Tsamardinos, Martha E. Pollack and John F. Horty, Merging plans with quantitative temporal constraints, temporally extended actions and conditional branches, Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS), 2000.
- [**Vossen et al 2000**] Thomas Vossen, Michael Ball, Amnon Lotem and Dana Nau, Applying integer programming to AI planning, Knowledge Engineering Review, 2000.
- [**Wolfman & Weld 1999**] Steven Wolfman and Daniel Weld, The LPSAT engine and its application to resource planning, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1999.