

International Journal on Artificial Intelligence Tools
 © World Scientific Publishing Company

T-SATPLAN: A SAT-based Temporal Planner

AMOL DATATRAYA MALI

*Electrical Engg. & Computer Science,
 University of Wisconsin, Milwaukee, WI 53211, United States of America
 mali@miller.cs.uwm.edu*

YING LIU

*Quad Graphics
 N63 W23075 State Hwy. 74
 Sussex, WI 53089, United States of America
 ying.liu@qg.com*

Received (1 March 2005)

Revised (14 February 2006)

Accepted (19 April 2006)

Recent advances in solving constraint satisfaction problems (CSPs) 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 planning problems in more expressive languages. These problems and languages contain metric time, quantifiers and resource quantities. SAT-based approaches are very effective at optimal planning. In this paper we report on SAT-based temporal planner T-SATPLAN. 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. In this paper we show how such a SAT encoding can be synthesized for temporal planning. As part of this, we generalize explanatory frame axioms for two states of fluents (true, false) to three states of fluents (true, false and undefined). We show how this SAT encoding can be simplified. We discuss two additional SAT encodings of temporal planning. The encoding schemes make it easier to exploit progress in SAT and CSP solving to solve temporal planning problems. We also report on an experimental evaluation of T-SATPLAN using one such encoding scheme. The evaluation shows that significantly large SAT encodings of temporal planning problems can be solved extremely fast.

Keywords: Temporal planning, Satisfiability, Encoding

1. Introduction

Recent advances in classical planning have made it possible to solve larger planning problems than before. Classical planning is the problem of finding an executable sequence of actions that achieves a partially specified goal state G from a completely specified initial state I , given the set of operators A . In classical planning, perception is assumed to be perfect, actions are assumed to be deterministic, and, environment is assumed to be completely observable and static. Some of the recently developed efficient classical planners cast planning as a CSP. Propositional

satisfiability (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^{1,2,3}, MEDIC⁴ and the planner which casts plan reuse and plan merging as satisfiability.⁵ 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.⁶ Advances in SAT solving like better branching heuristics and local search can be exploited to further improve the performance of SAT-based planners.

Planning domains described using languages more expressive than STRIPS contain quantifiers, conditional effects, metric time and resource quantities. Examples of such problems include many NASA planning applications.⁷ 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.^{7,8} The spacecraft has to gather scientific data by observing celestial objects with multiple instruments. These observations must be conducted within certain time windows since the targets are moving. Moving the observation instruments takes time and power. The on-board power is limited. Some actions have a variable duration.

There is an increasing interest in extending/adapting advances in classical planning to solving planning problems specified in more-expressive languages. Temporal Graphplan (TGP)⁷ is an extension of Graphplan⁹ for handling actions with durations. The LPSAT planner¹⁰ 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 part and a linear inequality, e.g. the constraint $((a > 3) \Rightarrow (x \vee y))$. LPSAT planner does not handle durative actions.

Development of planners handling more-expressive languages 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 less challenging for refinement planners. This is because refinement planners maintain different representations for different partial plans in the form of nodes in a search tree. A partial plan is a set of constraints which may be refined into a plan. In progression, an action sequence is executed starting at the initial state and it is checked whether goal is true at the end of its execution. 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 a partial-order planner contains a set of constraints like step-action bindings and partial orderings over the steps. The goal-achieving capability of such a partial plan can be verified simply by verifying that it is free of flaws like threatened causal links, open conditions and inconsistent step orderings. The verification of soundness and completeness of more expressive

planners which cast planning as a 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 constraint 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.

Though extremely fast heuristic search planners have been developed in the last nine years, SAT-based approaches continue to be effective for optimal planning. This is clear from the first place that SATPLAN04³, a SAT-based planner won in the optimal track of the classical part of the fourth international planning competition in 2004. The best approach for optimal parallel planning appears to be SAT or CSP, not heuristic search¹¹. Given this and the development of extremely fast SAT solvers, adapting SAT planning for handling durative actions is a worthwhile activity.

The adaptation of SAT planning to handle durative actions is challenging because of the task of identifying all constraints that are satisfied if and only if there is a plan of bounded length. Temporal planning under certainty 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 executable actions that have 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 assumptions⁷ in temporal planning when setting up the SAT encodings. Effects of actions are undefined during their execution. Action durations are fixed and are integers. Preconditions of an action should all be true at its start time. The effects of an action hold only at the end of its execution. The SAT encoding generated by T-SATPLAN is such that it has a solution if and only if there is a temporal plan whose execution time is at the most k time steps. T-SATPLAN is the first temporal planner to use SAT encodings. We identify nineteen 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 report on an empirical evaluation of T-SATPLAN that uses this encoding. The evaluation shows that many large temporal planning problems can be solved extremely fast. We also report on a comparison of T-SATPLAN with temporal planners TPSYS and TP4. We show how relaxed temporal planning graph can be used to get smaller encodings. We show how the encoding can be adapted to more complex case of temporal planning in which it is not necessary for different preconditions of an action to be true at the 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. We also show how our encodings can be extended to handle conditional effects and resource quantities.

The rest of the paper is organized as follows. In section 2, we explain the classical planning as SAT paradigm and the state-space SAT encoding with explanatory frame axioms for classical planning.² This encoding assumes that all actions are instantaneous. In section 3, we describe our state-space SAT encoding for temporal planning. In section 4, we discuss how the size of this encoding can be reduced and

how the encoding can be modified to handle a more complex case of temporal planning where different preconditions and/or effects of an action are true at different times. We also show how T-SATPLAN uses relaxed temporal planning graph to generate a smaller encoding. In this section we also show how temporal planning can be cast as a CSP other than SAT and describe two additional SAT encodings of temporal planning. One of these is the causal encoding whose size is not affected by durations of actions at all. We also review some work on satisfiability and temporal planning in this section. In section 5, we report on an empirical evaluation of T-SATPLAN. Conclusions are presented in section 6.

2. Background

In this section, we explain how SAT-based planners work and describe the state-space SAT encoding² for classical planning where all actions have a unit duration.

An action is a ground instance of an operator. For example, $move(x, y, z)$ is the operator for moving block x from top of block y to top of block z such that $x \neq y, y \neq z, x \neq z$. There are $O(n^3)$ actions obtained by grounding this operator when there are n blocks. Many classical planners use STRIPS representation. In this representation, a world state is a conjunction of function-free ground literals. An action has preconditions expressed as a conjunction of ground positive literals. All preconditions of an action must be true before it is applied. The effects of an action are expressed by a conjunction of ground literals which specify how a situation changes after the action is applied. $load(P_1, Plane_2, Heathrow)$ is the action of loading package P_1 in plane $Plane_2$ at *Heathrow* airport. Its preconditions are $package(P_1), plane(Plane_2), airport(Heathrow), at(P_1, Heathrow)$ and $at(Plane_2, Heathrow)$. Its effects are $in(P_1, Plane_2), \neg at(P_1, Heathrow)$. $in(P_1, Plane_2)$ is an add effect of the action and $at(P_1, Heathrow)$ is a delete effect of the action. \neg denotes negation. Predicates like $in(P_1, Plane_2)$ and $airport(Heathrow)$ can be treated as propositions simply by rewriting them as inP_1Plane_2 and $airportHeathrow$ respectively. In the rest of the paper, $\wedge, \vee, \Rightarrow$ denote logical AND, OR and implication respectively.

SAT-plan¹ works in the following manner: Based on the initial state, the number of steps assumed to exist in plan (k), goal state and the action descriptions, 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 at which actions can occur range from 0 to $(k - 1)$. The steps at which propositions can be true/false range from 0 to k . The encoding can be 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). If a satisfying assignment is found by a SAT solver, the solution (the satisfying truth assignment) is interpreted and the plan is output. If the encoding cannot be satisfied, the 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 or action variables.

The explanatory frame axiom-based state-space encoding² contains the following constraints: (i) All propositions true in the initial state are true at time 0 and all propositions false in the initial state are false at time 0. (ii) If an action occurs at time t , its preconditions 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 the 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 the truth of a proposition cannot change unless an action causing the change occurs. These constraints are known as explanatory frame axioms.

3. State-space SAT Encoding for Temporal Planning

To generate a SAT encoding, we first need to bound the number of steps in the plan. We denote this bound by k . O denotes the set of all actions. U denotes the set of all ground fluents in domain. A Boolean 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. $A(o_i)$ denotes the set of add effects of action o_i . $P(o_i)$ denotes the set of preconditions of o_i . $D(o_i)$ denotes the set of delete effects of o_i . Before explaining the constraints that 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 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 in the model of the encoding, it means that o_i starts at time t in the temporal plan. If $o_i(t)$ is assigned false, it means that o_i does not start at time t . d_i denotes the duration of the action o_i . The time steps or time points in the encoding at which actions may start range from 0 to $(k - d_{min})$. d_{min} is the minimum of the durations of all actions. d_{max} is the maximum of durations of all actions. The following constraints appear in our 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 preconditions.

6 *A. D. Mali & Y. Liu*

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 $(a \wedge b)$ and $\{a, b, c, d\}$ is the set of all ground fluents, the encoding contains $(at(0) \wedge bt(0) \wedge cf(0) \wedge df(0))$.
4. Fluents from goal are true at time k . If the 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 preconditions 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 $\max(|A(o_i)| + |P(o_i)| + |D(o_i)|)$. These constraints contribute $k \cdot |O|$ variables to the encoding. For example, if x and y are preconditions 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 points ranging from 0 to k .
6. If an action o_i does not delete any of its preconditions, 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 preconditions. These can be considered as temporal 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 preconditions and d' is the maximum of the durations of these actions.
7. If an action o_i does not delete its precondition x , then if o_i starts at time t , x remains true over the closed interval $[t + 1, t + d_i]$. If there are m actions that do not delete any of their own preconditions, these constraints contribute $O(m \cdot d_{max} \cdot k \cdot m')$ clauses to the encoding, where m' is the maximum number of preconditions of an action not deleted by the action, among all actions.
8. The preconditions 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 preconditions, these constraints contribute $O(m_1 \cdot d_{max} \cdot k \cdot m'')$ clauses to the encoding, where m'' is the maximum number of preconditions of an action deleted by the action, among all actions. For example, if the effects of action o_4 are $\neg p$ and q , such that p is one of its preconditions, and the 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 preconditions 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 preconditions deleted by it, among all actions. For example, if the effects of action o_4 are $\neg p$ and q , such that p is one of its preconditions, 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. As shown in Fig. 1, there are 6 kinds of changes that a state of a fluent may undergo.



Fig. 1. Six kinds of changes in the states of a fluent

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 with durations 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 2 and all possible effects two actions may have on a fluent p . The 7 cases in Fig. 2 are as follows. In (i), o'' starts when o' ends. In (ii), o'' and o' end at the same time and o'' starts after o' starts. In (iii), o' and o'' start and end at the same time. In (iv), o'' starts after o' starts and o'' ends before o' ends. In (v), o' and o'' start at same time but o'' ends after o' ends. In (vi), o'' starts after o' starts and before o' ends. o'' ends after o' ends. In (vii) o'' starts after o' ends.

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 . These 7 logical relations along with 7 possible temporal relations in Fig. 2 are used in arriving at the mutual exclusion constraints 16,17,18 and 19 next. Note that the previously mentioned constraints in our encoding avoid overlaps of certain conflicting actions. For example, a fluent cannot be true and false at the same time. A fluent cannot be false and undefined at the same time. A fluent cannot be true and undefined at the same time. Concurrency of actions that violates these constraints is automatically avoided by the presence of our previously mentioned constraints. So we do not include extra mutual exclusion constraints to handle these cases. Only those mutual exclusion cases that are not handled by the previously mentioned constraints are handled by constraints 16,17,18 and 19.

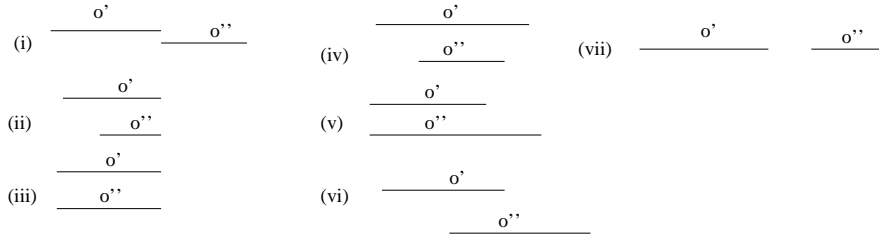


Fig. 2. Temporal relations between two actions o' and o''

16. If two actions o_i and o_j have the same duration such that o_i deletes some precondition 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 has precondition 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 $(t + d_i) = (t' + d_j)$, $t' \geq t$, $t \in [0, k - 1]$, $t' \in [0, k - 1]$. This constraint prevents two actions from making same fluent undefined at the same time.

18. 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 $(t + d_i) = t'$, $t \in [0, k - d_i]$, $t' \in [0, k - d_j]$.

19. 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 the ending time of o_j . This constraint prevents two actions from deleting the same fluent at the same time. 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 $(t + d_i) = (t' + d_j)$, $t \in [0, k - d_i]$, $t' \in [0, k - d_j]$.

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 at each time step. Once the encoding is solved, the temporal plan can be obtained by reading the truth assignment to the action variables from the model of the encoding. This truth assignment provides the starting times of all actions in the temporal plan along with the actions themselves.

T-SATPLAN generates encodings using a variation of this set of constraints. Its encodings do not have variables for undefined states of fluents. We show in section 4.4 that these variables can be eliminated. It uses a relaxed temporal planning graph (RTPG) to generate a smaller encoding. The notion of RTPG is explained in section 4.1.

4. Discussion

We showed how temporal planning can be encoded as SAT 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 preconditions of an action to be true at the same time and in which 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. We also show how the encoding can be adapted for handling conditional effects and resource quantities. We discuss some recent advances in SAT solving and SAT planning at the end of this section.

4.1. *Reducing the Encoding Size*

Though a smaller encoding is not always easier to solve, smaller encodings have been shown to be solvable faster.^{1,4,5} 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 graph⁹ can be used to wisely choose the value of k . Graphplan works in 2 phases. The first phase 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 from 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 preconditions 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 precondition 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 two kinds of edges in the plangraph: **(i)** edges from propositions in proposition level i to actions (whose precondition list contains these propositions) in action level i and **(ii)** edges from actions in action level i to propositions (which are add/delete 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 the preconditions and effects of the actions. These mutexes are permanent. Dynamic mutexes are found by propagating static mutexes using truths of conditions in the initial state. This propagation leads to even more dynamic mutexes that are propagated.

If the plangraph has a proposition level that contains all propositions from the

goal such that no two of these are mutex, Graphplan starts a backward search for a plan. 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 the 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 the last action level, (ii) the number of propositions in the last proposition level, (iii) the number of pairs of actions that are mutex in the last action level, and (iv) the number of pairs of mutex propositions in the last proposition level. In the planning graph in Fig. 3, 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 denote actions. The preconditions and effects of these actions are shown on the left and right sides of the boxes respectively, in the right part of Fig. 3. The plan found by Graphplan is a sequence of sets of actions. The goal is achievable with the plan Step 0: M2, Step 1: M1 & M3. This plan can be written as $[\{M_2\}, \{M_1, M_3\}]$. The actions M1 and M2 are statically mutex and the actions M3, M4 are dynamically 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.¹²

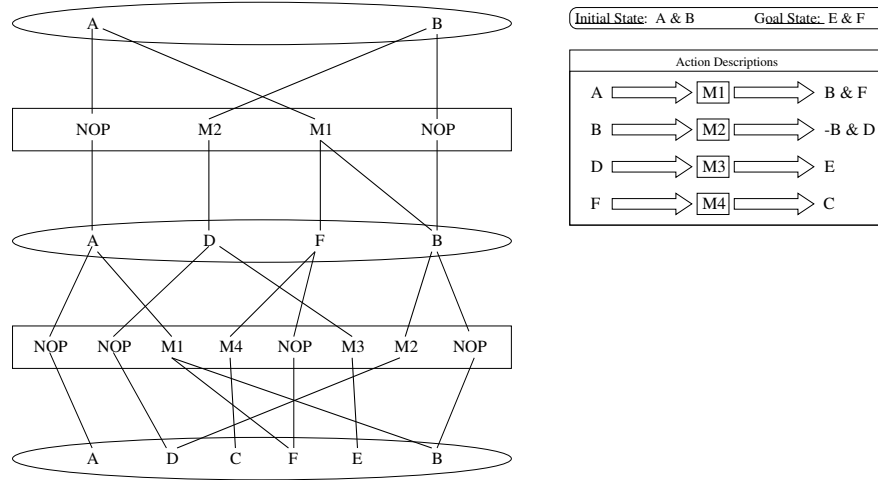


Fig. 3. A planning graph

It is clear that actions that do not appear in the 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 the no-ops from the action levels. This planning graph is constructed by Graphplan⁹ ignoring action durations. 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 SAT 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 time steps (same as time points) k in the SAT encoding can then be chosen so that $\theta' \leq k$ where $\theta' = (\min(d_2, d_4) + \min(d_2, d_4, d_5) + \min(d_2, d_4, d_5, d_8, d_9) + \min(d_2, d_4, d_5, d_8, d_9, d_{11}))$. Constructing a leveled off planning graph can take a significant amount of time, especially when $|O|$ is high.

One way to generate a smaller encoding is to use a relaxed temporal planning graph (RTPG). RTPG can be constructed much faster than the leveled off planning graph of Graphplan, though it does not give some information that leveled off planning graph does. This information is mutex relations. RTPG differs from planning graph of Graphplan in many ways. RTPG does not contain action levels and proposition levels. Delete effects of actions and numeric variables are ignored in the construction of RTPG. So RTPG construction does not involve computation and propagation of mutex relations between actions or propositions. A RTPG contains an action and proposition at the most once. The notion of RTPG was introduced by planner Sapa. Forward state-space heuristic search planner Sapa¹³ uses RTPG to find heuristic values of nodes in the search tree. A RTPG for a node is constructed by applying actions in the world state of that node in forward direction, ignoring delete effects of the actions and numerical preconditions and numeric effects. Only boolean preconditions and boolean add effects are used in constructing the RTPG. RTPG also includes starting and ending times of actions and times at which various propositions become true. The construction of RTPG in Sapa is terminated when all propositional subgoals from the goal of the planning problem are achieved in the RTPG. The RTPG constructed by T-SATPLAN is different from that in Sapa. RTPG is constructed by T-SATPLAN in a preprocessing phase to generate a smaller encoding. So T-SATPLAN constructs only one RTPG and Sapa constructs RTPG for every node in the search tree that is heuristically evaluated. T-SATPLAN constructs RTPG by applying actions in forward direction starting in initial state until no new actions can be applied. So the construction of RTPG in T-SATPLAN terminates when a different criterion than the one in Sapa is satisfied. So RTPG of T-SATPLAN generally contains more actions and more propositions than the RTPG of Sapa. RTPG of T-SATPLAN gives information like the following: a lower

bound on the earliest time point at which any action will occur in any temporal plan and a lower bound on the earliest time point at which any proposition will be made true by any temporal plan. The lower bounds for actions are same as the time points at which the actions start in the RTPG. The lower bounds for the propositions are same as the time points at which they become true in the RTPG. This information can be used to generate a smaller encoding in the following fashion. If the lower bound on the earliest time point at which action o_6 will start in any temporal plan is 6, then variables $o_6(0), o_6(1), o_6(2), o_6(3), o_6(4),$ and $o_6(5)$ need not be a part of the encoding. Similarly, if the time point at which proposition q appears in RTPG is 7, then variables $qu(0), qt(0), qu(1), qt(1), qu(2), qt(2), qu(3), qt(3), qu(4), qt(4), qu(5), qt(5),$ and $qu(6), qt(6)$ need not be created. Clauses involving these variables need not be created either. This RTPG-based size-reduction strategy is incorporated in T-SATPLAN. Considering the delete effects of actions and propagating mutual exclusion relations will give better lower bounds on the earliest start times of actions and earliest time points at which propositions become true, but this is not incorporated in T-SATPLAN. A relaxed temporal planning graph is shown in Fig. 4. Initial state $(a \wedge b \wedge c)$ holds at time 0. The goal $(f \wedge n)$ is achievable in a relaxed fashion at time 16. Action descriptions are in the form (preconditions) (action name) (effects). The graph shows that d, e cannot become true before time 3, f cannot become true before time 7, s, m cannot become true before time 10 and n cannot become true before time 16. The graph also shows that the lower bounds on the earliest times at which o_1, o_2, o_3, o_4, o_5 and o_6 can start in any temporal plan are respectively 0, 3, 0, 3, 10 and 0.

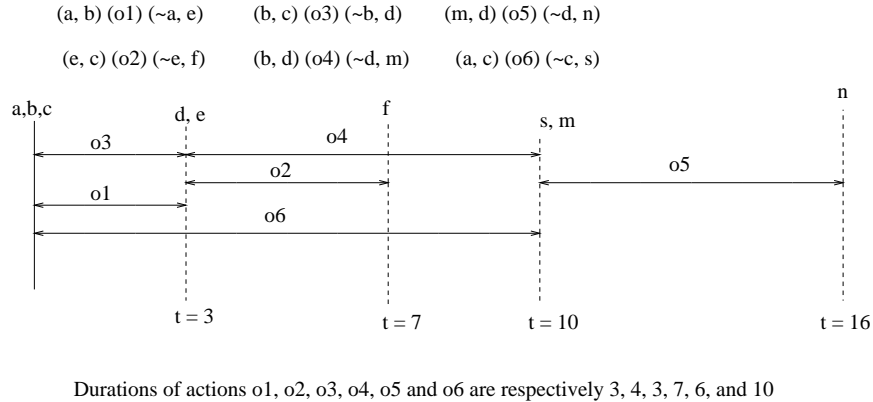


Fig. 4. Relaxed temporal planning graph.

4.2. *More Complex Temporal Planning*

We assumed that all preconditions 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 preconditions 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 preconditions are $clear(A)$, $on(A, B)$ and $clear(C)$. Let us assume that there are unlimited grippers and one does not need the precondition $hand - empty$. 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 at the start of $move(A, B, C)$ and the effect $on(A, C)$ becomes true much later, at its end. 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 preconditions are needed true relative to s and with the times at which 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 preconditions 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 precondition of an action deleted by itself is undefined over the interval $[t + 1, t + r - 1]$, when the precondition 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 adapted to handle such actions.

4.3. *Temporal Planning as CSP other than SAT*

Since SAT can be easily transformed into 0-1 ILP (integer linear programming problem), 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.

One can generate a CSP using the constraints that lead to our SAT encoding of T-SATPLAN. 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 other than SAT. Note that no extra effort is needed to verify the correctness of the CSP formulation.

4.3.1. Handling Conditional Effects and Resource Quantities

Conditional effects are context-dependent effects of actions. For example, consider the action of flying a plane from Heathrow airport to O'Hare airport. Precondition of the action is $at(Plane, Heathrow)$. Its add effect is $at(Plane, O'Hare)$ and its delete effect is $at(Plane, Heathrow)$. These are unconditional effects of the action. These effects are not context-dependent. The effect list also includes the following expression $\forall x(in(x, Plane) \rightarrow (at(x, O'Hare) \wedge \neg at(x, Heathrow)))$. This means that whatever is inside the plane is now at O'Hare and not at Heathrow. This universally quantified expression is the conditional effect expression. The effects $(at(x, O'Hare) \wedge \neg at(x, Heathrow))$ hold only in the context $in(x, Plane)$. Such actions can be handled by extending our encoding to handle conditional effects. Consider an action o_5 with duration 4 with effects $(a \wedge b \wedge (c \rightarrow e) \wedge ((p \wedge q) \rightarrow (f \wedge g)))$. e, f and g are conditional effects. Consider constraint 5 from section 3. The effects of the action can be encoded as follows: $(o_5(t) \rightarrow (a(t+4) \wedge b(t+4)))$, $((o_5(t) \wedge c(t)) \rightarrow e(t+4))$, $((o_5(t) \wedge p(t) \wedge q(t)) \rightarrow (f(t+4) \wedge g(t+4)))$. Other constraints from section 3 can be modified to handle conditional effects.

A mixed-integer-linear-programming encoding (MILP) for planning with resource quantities exists.¹⁴ In this work, Kautz and Walser handle actions that set resource quantity to its maximum value, actions that produce resources, and actions that consume resources. They extend Graphplan's notion of mutual exclusion actions based on interactions between boolean preconditions and effects to actions with numerical preconditions and numerical effects. Ideas from T-SATPLAN and

Kautz and Walser’s work can be used in order to develop an MILP encoding for temporal planning with production, setting and consumption of resources. A key extension needed to our encodings is the inclusion of numeric variables of form $v(t)$, to denote the quantity of resource v at time t . The new encoding will no longer be a purely SAT encoding due to the presence of numeric variables and $+$, $-$, $/$, $*$ operations.

4.4. *Other SAT Encodings*

In this subsection, we discuss two additional SAT encodings of temporal planning. The first is a variation of our state-space encoding with explanatory frame axioms for temporal planning. The other encoding is an augmentation of the causal encoding for classical planning.²

4.4.1. *Eliminating Certain Variables*

Let us revisit our state-space encoding with explanatory frame axioms for temporal planning from section 3. The variables for encoding undefined states of fluents can be eliminated from this encoding. The only use of these variables is that they prevent certain conflicting actions from overlapping. For example, if o_1 deletes p , p is undefined during the execution of o_1 . If o_2 needs p and does not delete it, then execution of o_2 and o_1 cannot overlap. Since p cannot be both true and undefined at the same time, executions of o_1 and o_2 cannot overlap. One does not need an explicit mutual exclusion constraint like $\neg(o_1(t) \wedge o_2(t))$ to prevent the overlapping of o_1 and o_2 when variables for undefined states are used. The variables for encoding undefined states of fluents can be eliminated and explicit mutual exclusion constraints can be added to the encoding. If the variables for encoding the undefined states of a fluent are eliminated, the variables for encoding true and false states can also be eliminated. So for a fluent p , one can eliminate $pt(t)$, $pf(t)$ and $pu(t)$ variables and just use $p(t)$ variable such that if $p(t) = true$, p is true at time t and if $p(t) = false$, p is false at time t . This elimination reduces the number of variables in the encoding. This allows us to remove the following types of clauses as well: $\neg(pt(t) \wedge pf(t))$, $\neg(pu(t) \wedge pf(t))$, $\neg(pt(t) \wedge pu(t))$. Instead of 6 kinds of explanatory frame axioms, only two kinds of explanatory frame axioms are needed, for change from *true* to *false* and vice versa, when the variables for undefined state are eliminated. This does however require an inclusion of several clauses for mutual exclusion between actions. This may not be a concern since the experiments on SAT encodings of planning show that reducing the number of variables is more important than reducing the number of clauses.

4.4.2. *Causal Encoding*

Let us consider the causal encoding for classical planning.² Though it has been shown¹⁵ that causal encodings are harder to solve than state-space encodings in

classical planning, causal encodings may be more useful for temporal planning than state-space encodings. This is because causal encodings represent partial order on steps and because there is no explicit representation of time. As a result if the difference in the durations of actions is high, this will not blow up the size of the causal encoding. Causal encodings represent all step-action bindings. For example, if there are 4 actions o_1, o_2, o_3 and o_4 and there are 3 steps p_1, p_2 and p_3 , the causal encoding contains the following constraints among many others: $((p_1 = o_1) \vee (p_1 = o_2) \vee (p_1 = o_3) \vee (p_1 = o_4) \vee (p_1 = \phi))$, $((p_2 = o_1) \vee (p_2 = o_2) \vee (p_2 = o_3) \vee (p_2 = o_4) \vee (p_2 = \phi))$, $((p_3 = o_1) \vee (p_3 = o_2) \vee (p_3 = o_3) \vee (p_3 = o_4) \vee (p_3 = \phi))$, $\neg((p_1 = o_1) \wedge (p_1 = \phi))$, $\neg((p_1 = o_2) \wedge (p_1 = \phi))$, $\neg((p_1 = o_3) \wedge (p_1 = \phi))$, $\neg((p_1 = o_4) \wedge (p_1 = \phi))$, $\neg((p_1 = o_1) \wedge (p_1 = o_2))$, $\neg((p_1 = o_1) \wedge (p_1 = o_3))$, $\neg((p_1 = o_2) \wedge (p_1 = o_3))$, $\neg((p_1 = o_2) \wedge (p_1 = o_4))$, $\neg((p_1 = o_1) \wedge (p_1 = o_4))$. In brief, it contains clauses that represent that each step is bound to one and only action, including no-op ϕ . ϕ has no preconditions. ϕ has no effects. The use of no-ops allows an extraction of plans with fewer steps than k . The causal encoding also contains clauses for resolving conflicts arising out of deletion of preconditions. It also contains clauses that specify antisymmetry, irreflexivity and transitivity of partial ordering over steps, along with some other clauses. The variables of type $p_i = o_j$ in the above clauses are boolean variables. If $p_i = o_j$ is assigned true, then step p_i is bound to action o_j . If $p_i = o_j$ is assigned false, then step p_i is not bound to action o_j . The causal encoding² has variables of type $p_i < p_j$. $<$ denotes partial order. If the boolean variable $p_i < p_j$ is assigned true, it means that step p_i precedes step p_j . This means that if action bindings of p_i and p_j are o_p and o_q respectively, then o_p occurs before o_q . Let us consider the case of temporal planning where (i) all effects of an action hold at the same time which is same as the end of its execution, and, (ii) all preconditions of an action must be true at same time which is same as the start of execution of the action. For this type of temporal planning, the causal encoding² can be used with the following changes: (i) This is about interpreting truth assignment of the partial-order variables. If $p_i < p_j$ is assigned true, the action bound to p_i ends before action bound to p_j starts. (ii) For each pair of actions o_i and o_j that cannot overlap, and, all pairs of steps p_a and p_b , the following clause should be added to the causal encoding²: $((p_a = o_i) \wedge (p_b = o_j)) \Rightarrow ((p_a < p_b) \vee (p_b < p_a))$. This leads to additional $O(k^2 \cdot |O|^2)$ clauses. k is the number of steps in the causal encoding. k does not represent the time duration of execution of plan. Once the encoding is solved, start times can be assigned to various actions in polynomial time using the truth assignments of the step-action binding variables (e.g. $p_a = o_i$) and the variables for partial orders on steps (e.g. $p_a < p_b$). That gives an executable temporal plan. Since the durations of actions and their start times are not represented in the encoding, the times do not affect the size of the encoding at all. Action durations do affect the size of the state-space encoding.

Like causal encoding, one can use state-space encoding without action durations for temporal planning. Note that one can solve the state-space encoding for STRIPS

planning problem obtained by ignoring action durations. One can derive a temporal plan from the STRIPS plan by inserting the action durations later. For example, let the STRIPS plan found by solving a state-space encoding with explanatory frame axioms be $[\{o_2, o_4\}, \{o_1, o_3, o_7\}, \{o_5\}]$. This plan can be converted into the following temporal plan $\{\langle o_2, 0 \rangle, \langle o_4, 0 \rangle, \langle o_1, \max(d_2, d_4) \rangle, \langle o_3, \max(d_2, d_4) \rangle, \langle o_7, \max(d_2, d_4) \rangle, \langle o_5, \max(d_2, d_4) + \max(d_1, d_3, d_7) \rangle\}$. The makespan of this plan is $(\max(d_2, d_4) + \max(d_1, d_3, d_7) + d_5)$. This may be much higher than the optimal makespan. Raising the number of steps in the state-space STRIPS encoding by 1 whenever the encoding is found to be unsatisfiable and inserting durations in the STRIPS plan does not guarantee a temporal plan with minimum makespan. Raising the number of steps in the state-space temporal encodings discussed in this paper by 1 and solving the encodings with a complete SAT solver generates a temporal plan with minimum makespan, if the initial value of k is a lower bound on the optimal makespan. So one can gain optimality by representing durations in the state-space encodings, as in section 3. Hence we reported on state-space encodings containing action durations.

4.5. SAT Solving

There has been an enormous progress in SAT solving in the last fifteen years. A very comprehensive survey of the SAT-related work over the period 1958-1996 exists.¹⁶ Some of the early local search algorithms for SAT solving are reported by Gu.¹⁷ A Lagrange-multiplier-based global search method for solving SAT instances has been developed.¹⁸ The method has very few algorithmic parameters to be tuned by users. Random walk, random walk with freebie move and Walksat are all local search methods for SAT solving.¹⁹ It is shown¹⁹ that it is beneficial to add implied clauses that capture certain long range variable dependencies and structure in the SAT instances.

4.6. Temporal Planning

TM-LPSAT²⁰ is a planner that can construct plans in domains containing atomic and durative actions, exogenous events and processes, discrete, real-valued and interval-valued fluents, reusable resources and continuous linear change to the quantities. It creates an encoding containing Boolean combinations of propositional atoms and linear constraints on numeric variables. However there are no experimental results demonstrating the efficiency of TM-LPSAT. A temporal planning problem can be formulated as a mixed-integer non-linear programming problem (MINLP).²¹ This paper also discusses a classification of 26 domain-specific and domain-independent planners assuming certainty, based on their state and temporal representations. The planners are classified into the following three categories: (i) Discrete time, discrete state, (ii) Discrete time, mixed state and (iii) Continuous time, mixed state. Mixed state planners handle Boolean as well as numeric variables. The constraints of a temporal planning problem can be partitioned by subgoals.²²

An encoding-based approach to generating a partially-ordered plan given a position-constrained plan exists.²³ A position-constrained plan specifies the exact start time for each action in the plan. Partially-ordered plans specify the relative orderings between the actions. Partially-ordered plans provide a better execution flexibility. The encodings²³ can be solved with a variety of objective functions, e.g. minimization of the number of pairs of actions that are ordered. A domain-independent approach to generating m subplans that collectively achieve the goal and that do not share any actions has been reported.²⁴ This is a SAT-based approach where an encoding is generated using two parameters. These are k (number of steps in each of the subplans and the global plan) and m , the number of subplans needed. The resulting subplans (if they exist) are generally fully-independent, as shown by results on Blocks world, Gripper, Meet-pass and Rocket domains. These subplans can be executed in parallel with flexibility. The ideas from the encoding of T-SATPLAN can be combined with the constraints in the independent-action encoding²⁴ to have an encoding whose model contains independent-action temporal subplans. Optimal parallel STRIPS plans are plans with minimum number of steps. These plans may have redundant actions. Optimal sequential STRIPS plans are plans with minimum number of actions. These plans lack parallelism however. A SAT-based approach to combine the advantages of parallel and sequential plans has been reported²⁵. This approach efficiently finds parallel plans with as few actions as possible.

5. Empirical Evaluation

We report on experimental evaluation of T-SATPLAN in this section. We also report on its experimental comparison with temporal planners TP4²⁶ and TPSYS²⁷ in this section. The SAT encoding generation code of T-SATPLAN is written in C++. Satz SAT solver from <http://www.intellektik.informatik.tu-darmstadt.de/SATLIB> was used to solve the SAT encodings. Satz is a complete SAT solver. The initial value of the number of steps in the encoding was equal to the sum of durations of operators, not actions. If encoding was not solved, the number of steps was increased by a user-given constant. The encodings generated by T-SATPLAN did not contain variables for undefined states of fluents. We showed at the start of section 4.4 that such variables can be eliminated. Hence the encodings used in the experiments did not contain the clauses containing variables for undefined states of fluents either.

Experimental results on T-SATPLAN are reported in Table 1. This information includes the makespans of the plans found and also the number of actions in the plans found. The makespan of a temporal plan is the positive difference between the maximum of the finish times of its actions and the minimum of the starting times of its actions. The solving times of encodings are reported in cpu seconds. The solving times do not include the times needed to detect the unsatisfiability of the previous encodings of the problem which had an insufficient number of steps. The solving times do not include operator-grounding times. The encodings were generated and solved on a 64 MHz machine with Linux Red-Hat 8.0. TP4 and TPSYS were also

ran on the same machine. 41 problems from 3 domains were used in the empirical evaluation. The domains used are Depots, Zeno travel and Driverlog. These domains were used in the third international planning competition in 2002. The Zeno travel domain we used did not have Zoom and Refuel operators and fuel levels. The number of planes in the Zeno travel problems varied from 1 to 3. The number of persons in these problems varied from 2 to 8. The number of cities in these problems were 3 or 4. The Driverlog problems had 1 or 2 drivers, 4 or 8 paths, 2 or 6 links, 2 packages, 1 or 2 trucks and 3 or 5 locations. The Depot problems had 1 depot, 1 or 2 distributors, 2 or 3 pallets, 1 to 3 crates, 2 or 3 hoists and 1 to 3 trucks. The number of variables in the encodings varied from 1770 to 14602. The number of clauses in the encodings varied from 8932 to 206812.

A model decoder to convert the satisfying truth assignment into a plan in English is also implemented by us in C++. We used this decoder to manually verify the correctness of the plans. All plans generated by T-SATPLAN were found to be correct.

We also compared T-SATPLAN with TP4 and TPSYS. TP4 is a heuristic search planner using iterative-deepening A* search algorithm with transposition tables for search control. TP4 does not use SAT encodings. TPSYS is a temporal planner based on Graphplan. There are several algorithmic differences between TPSYS and T-SATPLAN. TPSYS does not use SAT encodings. It is a variation of Graphplan for temporal planning. TPSYS propagates mutex relations and T-SATPLAN does not. TPSYS handles preconditions of actions needed at start, at end and over their full interval. TPSYS also handles add and delete effects of actions at their start and end.

TPSYS was run on 21 problems and TP4 was run on 41 problems. The number of actions in the plans of TP4 and T-SATPLAN was equal on 10 of the 41 problems. Makespans of the plans found by T-SATPLAN were always with a factor of 1.58 of the makespans of the plans found by TP4. On 23 problems, the makespans of the plans found by T-SATPLAN were within a factor of 1.1 of the makespans of the plans found by TP4. This is very encouraging since the makespans of temporal plans are generally more important in temporal planning than the plan generation times. The plan generation times are of the order of milliseconds, seconds, minutes or hours and the makespans are of the order of minutes, hours, days or months. So an increase in the plan generation time acceptable if that yields a large reduction in the makespan. T-SATPLAN found plans with lower makespans than TPSYS on 9 out of the 21 problems. Makespans of the plans found by T-SATPLAN were within a factor of 1.36 of the makespans of TPSYS on the remaining 12 problems. The makespans were equal on 3 out of these 12 problems. T-SATPLAN was faster than TPSYS on 4 problems and the speedup factors on these problems were 119.21, 1882.92, 807.94 and 1692.81.

One key difference between encoding-based approach like T-SATPLAN and non-encoding approach like TP4 and TPSYS is the ease of handling additional expressiveness features like conditional effects and resource quantities. Handling such fea-

Table 1. Empirical results obtained with T-SATPLAN

Problem	Solving time	# Actions	Makespan
ztravel-1	1.57	1	180
ztravel-2	16.15	24	60
ztravel-3	1.38	14	47
ztravel-4	2764.56	25	80
ztravel-5	1.31	9	25
ztravel-6	0.88	18	28
ztravel-7	2.94	17	50
ztravel-8	0.46	6	45
ztravel-9	4.37	8	85
ztravel-10	0.56	7	47
depot-1	0.1	5	19
depot-2	0.17	6	19
depot-3	0.52	10	20
depot-4	12.37	32	30
depot-5	1.18	9	20
depot-6	2.75	10	20
depot-7	27347.7	26	30
depot-8	2.75	10	20
depot-9	0.46	30	20
depot-10	0.97	15	19
depot-11	3.77	43	30
depot-12	27715.07	55	40
depot-13	2.02	21	20
depot-14	9.09	18	23
depot-15	0.77	15	19
depot-16	2.94	21	25
dlog-1	0.62	11	15
dlog-2	1.49	27	15
dlog-3	1.9	26	20
dlog-4	0.71	42	40
dlog-5	1.08	26	55
dlog-6	0.42	26	55
dlog-7	0.54	8	55
dlog-8	1.25	12	75
dlog-9	0.57	17	80
dlog-10	7.59	36	90
dlog-11	1.46	17	80
dlog-12	5.08	20	50
dlog-13	0.18	25	20
dlog-14	3.5	19	50
dlog-15	0.56	42	70

tures with T-SATPLAN approach means encoding extra constraints and finding an appropriate solver for these constraints, e.g. CPLEX solver for MILP problems. Handling such features in TP4 and TPSYS is more complex since changes to the planning algorithms are needed.

6. Conclusion

SAT-based planners are extremely effective at generating optimal STRIPS plans. This and the enormous progress in SAT solving makes it worthwhile to examine the utility of SAT approach for solving problems in planning languages more expressive than STRIPS. 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 temporal plan whose makespan is less than or equal to the chosen bound k . A key idea in the development of this encoding is explanatory frame axioms that handle three states of propositions (true, false and undefined) rather than two states (true and false). We showed how the size of the encoding can be significantly reduced using information from the planning graph. We reported on temporal planner T-SATPLAN which generates state-space encodings of temporal planning and solves them using complete SAT solver Satz. T-SATPLAN uses relaxed temporal planning graph to generate smaller encodings. Our empirical evaluation verified the correctness of T-SATPLAN. Our experimental comparison of T-SATPLAN with planners TP4 and TPSYS showed that makespans of the plans found by T-SATPLAN are close to those of plans found by TPSYS and TP4. The evaluation of T-SATPLAN shows that many large SAT encodings of temporal planning can be solved significantly fast. We showed how the SAT encoding can be adapted to handle actions all preconditions of which need not be true at the 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. We also showed how our encodings can be adapted to handle resource quantities and conditional effects of actions. We also reported on two additional encodings for temporal planning. One of them is causal encoding whose size is not affected by durations of actions at all.

Acknowledgement: This work was funded by NSF grant IIS-0119630 to the first author. We thank Subbarao Kambhampati, David Smith, Martha Pollack, Maria Fox, Tan Son, Chitra Kashikar and Ronen Brafman for their useful comments on the ideas in this paper. We thank Antonio Garrido for making planner TPSYS available to us. We thank Patrik Haslum for answering questions about planner TP4. This work was performed when Ying Liu was a graduate student in the computer science program at University of Wisconsin, Milwaukee.

References

1. Henry Kautz and Bart Selman, Pushing the envelope: Planning, propositional logic and stochastic search, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1996, pp. 1194-1201.
2. Henry Kautz, David McAllester and Bart Selman, Encoding plans in propositional logic, *Proceedings of Knowledge Representation and Reasoning conference (KR)*,

- 1996, pp. 374-384.
3. Henry Kautz, SATPLAN04: Planning as Satisfiability, *International planning competition*, Whistler, Canada, Edited by Stefan Edelkamp, Joerg Hoffmann, Michael Littman and Hakan Younes, 2004, pp. 44-45
 4. Michael Ernst, Todd Millstein and Daniel Weld, Automatic SAT compilation of planning problems, *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1997, pp. 1169-1177.
 5. Amol Mali, Plan merging and plan reuse as satisfiability, *Proceedings of European Conference on Planning (ECP)*, Durham, UK, 1999, pp. 84-96.
 6. Ronen I. Brafman, A simplifier for propositional formulas with many binary clauses, *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 515-522.
 7. David E. Smith and Daniel S. Weld, Temporal planning with mutual exclusion reasoning, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999, pp. 326-337.
 8. David E. Smith, Coping with time and continuous quantities, Invited talk at the *International Conference on Artificial Intelligence Planning Systems (AIPS)*, 2000.
 9. Avrim Blum and Merrick Furst, Fast planning through planning graph analysis, *Artificial Intelligence* **90**, 1997, pp. 281-300.
 10. Steven Wolfman and Daniel Weld, The LPSAT engine and its application to resource planning, *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1999, pp. 310-317.
 11. Hector Geffner, Heuristic search planning: Progress and challenges, Slide 14 of the slides of the Invited talk at European Conference on planning (ECP), 2001.
 12. Henry Kautz and Bart Selman, Unifying SAT-based and graph-based planning, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999, pp. 318-325.
 13. Minh B. Do and Subbarao Kambhampati, Sapa: A domain-independent heuristic metric temporal planner, *Proceedings of the European Conference on Planning (ECP)*, 2001.
 14. Henry Kautz and Joachim Walser, State-space planning by integer optimization, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999, pp. 526-533.
 15. Amol Mali and Subbarao Kambhampati, On the utility of causal encodings, *Proceedings of the national conference on artificial intelligence (AAAI)*, 1999, pp. 557-563.
 16. Jun Gu, Paul W. Purdom, John Franco and Benjamin W. Wah, Algorithms for the Satisfiability (SAT) problem: A survey, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **35**, 1997, pp. 19-151
 17. Jun Gu, Local search for satisfiability (SAT) problem, *IEEE Transactions on Systems, Man and Cybernetics* **23**, No. 4, July/August 1993, pp. 1108-1129
 18. Benjamin Wah and Yi Shang, A discrete Lagrangian-based global-search method for solving satisfiability problems, *DIMACS Series in Discrete Mathematics and*

Theoretical Computer Science **35**, 1997, pp. 365-392

19. Wei Wei and Bart Selman, Accelerating random walks, *Proceedings of the 8th international conference on principles and practices of constraint programming (CP)*, 2002, pp. 216-232
20. Ji-Ae Shin and Ernest Davis, Processes and continuous change in a SAT-based planner, *Artificial Intelligence* **166**, No. 1-2, 2005, pp. 194-253
21. Benjamin W. Wah and Yixin Chen, Constraint partitioning in penalty formulations for solving temporal planning problems, *Artificial Intelligence* **170**, Issue 3, March 2006, pp. 187-231
22. Benjamin W. Wah and Yixin Chen, Subgoal partitioning and global search for solving temporal planning problems in mixed space, *International Journal on Artificial Intelligence Tools* **13**, No. 4, 2004, pp. 767-790
23. Minh B. Do and Subbarao Kambhampati, Improving the temporal flexibility of position-constrained metric temporal plans, *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2003, pp. 42-51
24. Amol Dattatraya Mali, SAT-based synthesis of independent subplans for a parallel and flexible execution, Technical report, Computer science, University of Wisconsin, Milwaukee, February 2006, pp. 1-6.
25. Markus Buttner and Jussi Rintanen, Satisfiability planning with constraints on the number of actions, *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005, pp. 292-299
26. Patrik Haslum and Hector Geffner, Heuristic planning with time and resources, *Proceedings of European Conference on Planning (ECP)*, 2001.
27. Antonio Garrido, A temporal planning system for level 3 durative actions of PDDL 2.1, *Proceedings of workshop "Planning for temporal domains"*, Toulouse, April 2002, pp. 56-66