Planning As CSP : A Transition Based Encoding

Student: Debdeep Banerjee  
RSISE-ANU, NICTA  
debdeep.banerjee@rsise.anu.edu.au

Supervisor: Patrik Haslum  
NICTA, RSISE-ANU  
patrik.haslum@anu.com.au

Background and Motivation

Domain independent automated planning is hard. It has been shown that classical planning is PSPACE complete (Bylander 1994). But the bounded planning problem is in NP. Here bounded means restricting the planning problem by some means, such as plan length (number of actions in the solution) or the number of time steps (makespan). Kautz and Selman (1992) were the first to describe a polynomial reduction of a bounded planning problem to a SAT problem, which is also in NP. Since then the bounded planning problem has been compiled to other substrates such as CSP ((Do & Kambhampati 2001), (Vidal & Geffner 2006)) and IP (Vossen et al. 1999). Most of these compilation approaches are based on the bounded horizon encoding, where a problem is bounded by the plan length or the makespan.

The main problem with the bounded horizon encoding is that it is hard to guess an initial bound which is close to the solution. Usually the initial encoding is bounded by a lower bound. If it doesn’t have a solution then the bound is increased by a minimum amount and this process is repeated until the encoding has a solution. This approach is not very efficient for problems where the estimated lower bound is too far from the solution. So two questions remain: “Can we bound a problem, other than bounding the length or makespan, such that the initial encoding will likely to have a solution?” and “If the initial encoding fails, can we choose the next bound more intelligently?” By intelligently, we mean guessing the next bound, which will increase the chance of having a solution, through analyzing the recent failures or the problem structures.

Here we propose a technique for compiling a bounded planning problem into a CSP, which aims to answer the first question and set a direction for the second. In the proposed approach we construct a CSP from the Transition Based Encoding of the planning problem which is bounded by the number of times each action can occur in the plan. Initially we start with a bound of 1, which implies that each action can occur at most once in the plan. Many planning problems can be solved using each action at most once (Vidal & Geffner 2006). So the initial CSP encoding has a high chance of being solvable. If the initial CSP has no solution, then we believe that it may be possible to find the set of actions that are needed more than once. We can allow only these actions to appear more than once, instead of increasing the bound for all actions.

Proposed Encoding

We encode a multi-valued state variable representation of a planning problem (a.k.a SAS+ (Bäckström & Nebel 1995)) into a transition based encoding as described below. This encoding is based on the ideas of Tuan A. Nguyen and Minh Do.

Transitions

Actions cause state variables to change from one value to another. A transition, in the proposed encoding, represents how an action changes a state variable’s value from one to the other. We use the following notation to represent a transition.

\[ T^{V}_{f \rightarrow g} \]

Here, \( V \) is the variable whose value is changed, and \( a \) is the action that changes the value from \( f \) to \( g \). The variable-value pairs \( V = f \) and \( V = g \) are the precondition and postcondition of the transition, respectively. A transition \( T \) can support a transition \( T_2 \), if \( T \)’s postcondition matches the precondition of \( T_2 \). We distinguish between two types of transitions: \( PREVAIL \) transitions that don’t change the value of the variable (preval conditions of the corresponding action) and \( EFFECT \) transitions that change variables’ value (effects of the corresponding action). In the transition based encoding each action corresponds to a unique set of transitions. For example, if action \( a \) has the prevail condition \( v_1 = x \), the precondition \( v_2 = z \) and the postcondition \( v_2 = w \), then the action \( a \) corresponds to the set \( \{ T^{v_1}_{x \rightarrow z}, T^{v_2}_{z \rightarrow x}, T^{v_2}_{z \rightarrow w} \} \). Similarly each transition corresponds to an unique action.

In the transition based encoding we assume each plan starts with a dummy \( START \) action and finishes with a dummy \( FINISH \) action. The \( START \) action has no preconditions and its effects are the initial values of the state variables. The \( FINISH \) action has the goal conditions as its preconditions and has no effect. The transitions of the \( START \)

\[ ^3 \]This is similar to the plan constructs in partial order planning.
action are called INITIAL transitions and the transitions of the FINISH action are called GOAL transitions.

CSP Formulation

In the transition based encoding a planning problem is represented as a set of transitions and their relation with the actions. If one transition supports another, this implies an ordering between the actions corresponding to those transitions. Finding a plan (as a sequence of actions) is equivalent to the problem of finding which transitions are in the plan and how they are supported.

We formulate this problem as a CSP, where each transition $T_{f\rightarrow g}^v$ (except for the INITIAL transitions), corresponds to a CSP variable $C[T_{f\rightarrow g}^v]$. The domain of this variable contains the transitions that can support $T_{f\rightarrow g}^v$. Since PREVAIL transitions don’t change any state variable’s value, only EFFECT transitions are used as domain values. The INITIAL transitions represent the initial state of the planning problem, so all others will be in the plan and don’t need to be supported. For this reason, the INITIAL transitions don’t correspond to any CSP variable but appear in some transitions’ domain. Except for the GOAL transitions, all the transitions have a dummy value, ⊥ (not-in-plan), in their domain. If ⊥ is assigned to a transition, it means that the transition is not in the plan.

Constraints are defined based on the causal relationship between transitions and actions.

1. Transition Activation Constraints: When a transition (value) is assigned to another transition (variable), it means both the transitions are in the plan and the value transition supports the variable transition. Each variable-value assignment implies that both the actions (corresponding to the variable and values transitions) are in the plan. That means all the other transitions caused by these actions are also in the plan, and therefore the not-in-plan value (⊥) must be removed from their domains.

   If $C[T_{f\rightarrow g}^v] = T_{f\rightarrow g}^v$ then
   \[
   \forall T \in \{\text{Transitions of } a1 - T_{f\rightarrow g}^v\} \quad C[T] \neq ⊥
   \]
   \[
   \forall T' \in \{\text{Transitions of } a2 - T_{f\rightarrow g}^v\} \quad C[T'] \neq ⊥
   \]

2. Transition Deactivation Constraints: If a transition is assigned the value ⊥, that means the transition is not in the plan, so all other related transitions caused by the same action must be excluded from the final plan.

   If $C[T_{f\rightarrow g}^v] = ⊥$ then
   \[
   \forall T \in \{\text{Transitions of } a\} - T_{f\rightarrow g}^v
   \]

   $C[T] = ⊥$

3. Cycle Check Constraints: In the transition based encoding, a variable-value assignment implies an ordering between the two transitions. Consequently, an ordering between two transitions implies the ordering between the corresponding actions as described in the following:

   \[
   C[T_{f\rightarrow g}^v] = T_{f\rightarrow g}^v \quad \Rightarrow \quad T_{f\rightarrow g}^a \text{ supports } T_{f'\rightarrow g'}^v
   \]
   \[
   a1 \text{ supports } a2 \text{ means } a2 \text{ must come before } a1 \text{ in the plan.}
   \]

   A valid plan can not have such a cycle. After each assignment, the cycle check constraint (CCC) removes the possible cycles proactively. The following example describes the working of the CCC.

   Suppose in some point in the search, $T_{f\rightarrow g}^v$ is assigned to $C[T_{f\rightarrow g}^v]$. In the following figure, the dotted rectangle represents the partial order due the assignment and the dotted arrows represent some of the possible cycles with respect to the commitments made by the search so far.

   For the assignment CCC finds two sets of transitions as described in the figure below. The Supported Transition Set contains the transitions caused by $a1$ and the actions that are transitively supported by $a1$, and the Supporting Transition Set contains the transitions caused by $a2$ and the actions that transitively support $a2$.

   To eliminate all possibilities of creating a cycle with respect to the ordering implied by the recent variable-value assignment, the transitions in the supported transition set are removed from the domain of each transition in the supporting transition set.

4. Unique Support: No two EFFECT transitions are supported by the same EFFECT transition.
For $C[T' \leftarrow f \leftarrow g] = T' \leftarrow f \leftarrow g$  

If $T' \leftarrow f \leftarrow g \in \{\text{EFFECT Transitions}\}$  
then $\forall T \in \{\{\text{EFFECT Transitions}\} \setminus T' \leftarrow f \leftarrow g\}$  

$C[T] \neq T' \leftarrow f \leftarrow g$  

However more than one PREVAIL transitions can be supported by the same EFFECT transition.

5. Implied Support Constraints: If an EFFECT transition $T_S$ supports a set of PREVAIL transitions $\{T_{P1}, T_{P2}, \ldots, T_{Pn}\}$ and an EFFECT transitions $T_E$, then each PREVAIL transition, in the supported prevail set of $T_S$, also supports $T_E$.

CSP Search

The search is guided by an Active Transition List (ATL) $^3$, which contains the transitions that need to be supported to achieve the goals. Initially the ATL contains the GOAL transitions (since each plan ends with the FINISH action). In each search step we pick a transition from the ATL and assign a support for it. From each assignment we can infer, via the transition activation constraint, the transitions that need to be supported and add them into the ATL. If all transitions in the ATL are assigned (supported), then we know that we have achieved the goals, and all remaining unassigned transitions can be set to $\perp$. The following section describes the variable and value selection heuristics for the search.

Heuristics

We have experimented with three different variable and value selection heuristics as described below.

CSP-Based Heuristic

This heuristic is the standard “fail-first” or “min domain size” heuristic for CSPs. It aims to choose the most constrained variable, and the value that will increase the active transition list by the least amount. The heuristic picks the transition with the smallest domain size from the active transition list as the next variable to assign. We rank the transitions from its domain by their number of related transitions and pick the transition with the lowest related transition size as its support. The number of related transitions of a transition is the number of transitions (not supported yet) caused by the same corresponding action, minus one.

Distance-Based Heuristic

This heuristic makes use of the distance-based planning heuristic, $h^2$ (Haslum & Geffner 2000). For each transition we calculate the $h^2$ value for achieving its precondition. We rank transitions in the active transition list by their $h^2$ values and pick the transition with the highest value as the next variable to assign. We rank the domain of the selected transition by their $h^2$ values and pick the transition with the lowest value. In case of a tie, transitions that achieve active nodes of their corresponding DTGs are given higher preference. Active nodes of a DTG are the values of the corresponding variable, that are achieved by some transitions in the active transition list. The intuition behind this heuristic is that we choose the transition farthest from the initial state as the next variable to assign and choose a transition closest to the initial state among the possible supports for the transition selected as variable as its value.

Hybrid Heuristic

This heuristic uses the same variable selection as the CSP-based heuristic and selects value of the variable using the distance-based heuristic as described above.

Initial Results and Future Work

Initial experiments have shown that the CSP-based heuristic and hybrid heuristic perform better than the distance-based heuristic. The distance-based heuristic was able to solve the first 5 problems of the Rover problem set from IPC3, while the CSP-based and hybrid heuristics solved the first 9 and 10 problems respectively. For problems (such as the problem 6 and 9 in the rover problem set) in which an action must occur more than once in order to increase the bound appropriately.

References


---

$^3$This is similar to the open precondition list in partial order planning.