On Realizing Planning Programs in Domains with Dead-end States

Federico Falcone and **Alfonso E. Gerevini** and **Alessandro Saetti**
Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Brescia, Italy
name.surname@unibs.it

Abstract

Agent planning programs are finite-state programs, possibly containing loops, whose atomic instructions consist of a guard, a maintenance goal, and an achievement goal, which act as precondition-invariance-postcondition assertions in program specification. The execution of such programs requires generating plans that meet the goals specified in the atomic instructions, while respecting the program control flow. Recently, De Giacomo *et al.* (2016) presented a technique, based on iteratively solving classical planning problems with action costs, for realizing planning programs in deterministic domains. Such a technique works generally well for domains with no or very few dead-end states. In this paper, we propose an enhancement of this technique to handle deterministic domains that have potentially many dead-end states, and we study the effectiveness of our technique through an experimental analysis.

Introduction

Agent planning programs are finite-state programs, possibly containing loops, whose atomic instructions consist of a *guard*, a *maintenance goal*, and an *achievement goal*, which act as classical Computer Science *precondition-invariance-postcondition* assertions (De Giacomo *et al.* 2016). The execution of an agent planning program requires generating plans that meet the goals specified in the atomic instructions, while respecting the program control flow. In particular, the generated plans should not block further plans needed to execute future instructions.

In a planning program, the dynamics of the world is described with a planning domain and an initial state, as usually done in planning. On top of such a domain, an agent planning program is modeled as a transition system, typically including loops, in which states represent *choice points* and transitions specify possible courses of actions that the agent may decide to follow. Such transitions constitute the high-level actions available to the agent, and are characterized by: a guard, which poses executability conditions in terms of the state of the domain; a maintenance goal, which specifies invariants that are guaranteed to hold for the course of actions to execute; and an achievement goal, which specifies the postcondition that the transition will achieve.

Intuitively, agent planning programs are meant to work as follows: at any point in time, the domain and the program are in some state, and the agent decides, autonomously, which program transition, among those whose guards are satisfied in the current state, to request. A synthesized plan for the transition goals is then executed, thus moving the domain and the program to their next states, from which a new request can be issued, a new plan executed again, and so on. Once a plan is associated to each request, at each point in time, we say that the agent planning program is "realized". Importantly, in synthesizing the plan for a transition, we need to take into account that the resulting state of the domain, not only must satisfy the achievement goal, but must also allow for the existence of plans for each possible next transition, and this must hold again after such plans, and so on ad infinitum.

As discussed by De Giacomo *et al.* (2016), the work on agent planning programs is related to generalized planning, in the sense that the result of the planning program realization can be seen as a form of generalized plan (e.g., (Bonet, Palacios, & Geffner 2009; De Giacomo *et al.* 2010; Srivastava, Immerman, & Zilberstein 2011)). Planning programs can also be considered as a form of complex routines, modelling desired domain evolutions and typically including conditions and cycles, that an agent executes in the domain. In planning, similar routines can be specified by temporally extended goals (e.g., (Bacchus & Kabanza 2000; Baier, Bacchus, & McIlraith 2009; De Giacomo & Vardi 1999; Gerevini *et al.* 2009; Kabanza & Thiébaux 2005)).

De Giacomo *et al.* (2016) propose an effective approach for realizing a planning program in deterministic domains. This approach is based on exploiting classical planning, and the specific algorithm that is developed and experimented works generally well for domains with few or no dead-end states. A dead end is a state from which the goal cannot be reached. When the domain includes no dead-end states, the computation of a realization of the planning program can be decomposed into the computation of a realization for every individual program transition. Such a decomposed computation is viable because, in domains with no dead-end state, the realization of a program transition incoming to a program state v does not compromise the realizability of the program transitions outgoing from v. On the contrary, when a domain contains dead-end states, the way by which a program transition is realized can affect the realizability of the successive transitions.

In this paper, we propose an enhancement of the realization techniques described in (Gerevini, Patrizi, & Saetti 2011; De Giacomo *et al.* 2016) to effectively deal with dead-end states. The proposed techniques are still based on

the usage of classical planning. Specifically, first we define a multiple planning problem with preferred and forbidden end-states as a sequence of planning problems such that the solution plan of each of these problems realizes a program transition, does not end into a forbidden state, and possibly ends into a preferred state. Then, we extend the algorithm proposed in (De Giacomo *et al.* 2016) to solve multiple planning problems with preferred and forbidden end-states. This algorithm uses a scheme for translating this special class of planning problems into classical planning problems with action costs. Finally, we evaluate the effectiveness of the proposed enhanced technique.

Agent Planning Programs

Agent Planning Programs (planning programs, or *p*-programs for short) are high-level representations of the behavior of agents acting in a domain (De Giacomo *et al.* 2016). Essentially, they are transition systems, with states representing decision points, and transitions, labelled by triples consisting of a guard, a maintenance goal and an achievement goal over the domain, representing atomic instructions of programs. For instance, a very simple planning program for a traveller routine is depicted in Figure 1, under which the agent (i.e., the traveller) continuously travels back and forth between New York and London.

Informally, in order for a planning program to be executable, each transition goal requires a plan to bring it about. Moreover, those plans ought to be "synchronized" so that the final world state generated by each plan is a suitable initial state for the subsequent plans associated with the next goals. When this is the case, the planning program is *realized*. In general, however, computing a realization does not simply amount to matching program transitions with appropriate plans. In fact, as plans are executed, both the state of the planning program and that of the underlying domain evolve and, in general, the planning program may reach the same state in different domain states, so that there is no guarantee that a single plan would work in all such domain states. Thus, a more sophisticated solution concept is required.

We deal with a specialization of the planning program realization problem (De Giacomo *et al.* 2016) by assuming a deterministic underlying planning domain. Formally, a *planning program for a deterministic planning domain* $\mathcal{D}$ is a tuple $\mathcal{P} = \langle A, P, V, v_0, \delta \rangle$, where:

- A is a finite set of *actions* of $\mathcal{D}$;
- P is a finite set of *propositions* of $\mathcal{D}$;
- V is the finite set of *program states*;
- $v_0 \in V$ is the *program initial state* of $\mathcal{P}$; and
- $\delta \subseteq V \times \Phi(P) \times \Phi(P) \times \Phi(P) \times V$ is the *transition relation* of $\mathcal{P}$, where $\Phi(P)$ stands for the set of all boolean formulas built from the set of propositions P. A transition $\langle v, \langle \gamma, \psi, \phi \rangle, v' \rangle \in \delta$ is used to denote that whenever the *guard* γ holds (in the domain), the agent planning program $\mathcal{P}$ may legally move from state v to state v' by "*achieving* ϕ *while maintaining* ψ."

A domain action is represented as a triple $\langle$*Pre*, *Eff*$^+$, *Eff*$^-\rangle$ where *Pre* is a set of boolean formulas representing the action preconditions, and *Eff*$^{+/-}$ is a set of propositions representing the action positive/negative effects. Like in classical planning, under the closed world assumption, a $\mathcal{D}$-state is specified by a set of propositions, an action $a = \langle$*Pre*, *Eff*$^+$, *Eff*$^-\rangle$ is said to be *executable* in a domain state s if $s \models$*Pre*, and the domain state s' obtained by executing a in state s is $s \setminus$ *Eff*$^- \cup$ *Eff*$^+$.

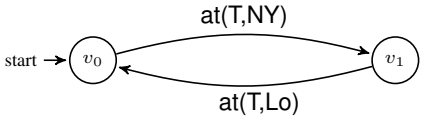

Figure 1: A very simple example of (the transition graph of) a planning program. Each edge is annotated with its corresponding achievement goal. For simplicity, maintenance goals and guards are omitted (they are "true" conditions).

Notation $Last(\pi(s))$ refers to the final $\mathcal{D}$-state obtained upon executing plan π from s (written $\pi(s)$). We say that from s a plan π *achieves* a goal ϕ, i.e., a propositional formula over the propositions of $\mathcal{D}$, if $Last(\pi(s)) \models \phi$, where satisfaction is defined as usual in propositional logic. Similarly, we say that from s a plan π *maintains* a goal ψ, if $s \models \psi$ and $s'' \models \psi$, for every intermediate state s'' generated by executing π from s.

Example 1 *The traveller* T *wants to continuously travel from New York (*NY*) to London (*Lo*) and from* Lo *to* NY*. The traveller moves among these cities by airplane* A*, which has to be periodically refueled. Assume that* A *can be refueled only at the headquarter of its airline, say Paris (*Pa*); and* A *has a fuel tank, which can be full (*FL2*), half full (*FL1*), or empty (*FL0*). Moreover, assume that initially* T *and* A *are at* Pa*, and the fuel level of* A *is* FL2*. The domain actions are* Board(x)*,* Debark(x)*,* Refill(x)*, and* Fly(x,y,z,w)*, which respectively represent that, at city x, the traveller can board to and debark from* A*, the fuel level of* A *can be increased from the current level x to* FL2*, and* A *can fly from city x to city y while the fuel in its tank changes from level w to level z. A graphical representation of the traveller's behavior is provided in Figure 1, where the transition system represents a planning program for* T*.*

When the planning program and the domain are in states v and s (initially v_0 and s_0), respectively, the agent is allowed to *choose* any enabled (i.e., whose guard holds true in s) planning program transition $\langle v, \langle \gamma, \psi, \phi \rangle, v' \rangle$ in $\mathcal{P}$. Note that for the simple planning program depicted in Figure 1, in practice, the agent does not choose any transition, since for both v_0 and v_1 the number of outgoing program transition is 1. If the planning program had an additional program state, say v_2, and an additional program transition from v_0 to v_2, then a transition selection would take place every time the program state is v_0.

Being declarative assertions, the chosen transitions are *not* directly executable and actual *realizations* are required for them. A realization, then, must provide a concrete plan π that brings about the achievement goal ϕ while guaranteeing maintenance of ψ and, furthermore, be compatible with further realizations for subsequent transitions of the planning program. The latter requirement is central to the approach, as the choice points in the planning program are resolved by decisions made by the agent *only at runtime*.

The notion of planning program realization is based on the following notion of *simulation*. A *simulation relation* is a relation $R \subseteq V \times 2^P$ such that $\langle v, s\rangle \in R$ implies that, for every transition $\langle v, \langle \gamma, \psi, \phi\rangle, v'\rangle$ in $\mathcal{P}$ such that $s \models \gamma$, there exists a plan π such that: π *achieves* ϕ and *maintains* ψ from s (in which case, plan π is said to *realize* the transition), and $\langle v', Last(\pi(s))\rangle \in R$. Note that an adequate plan for a transition, i.e., one that realizes the transition while preserving a simulation relation for the *p*-program, might not be the shortest one that reaches the achievement goal while maintaining the maintenance goal of the transition. Indeed, such a plan may actually prevent future agent requests (*p*-program transition choices) from being fulfillable.

Let 2^π be the set of plans in the planning domain. A *realization* of an agent planning program $\mathcal{P}$ in planning domain $\mathcal{D}$ from an initial $\mathcal{D}$-state $s_0 \in 2^P$ is a partial function $\Omega : 2^P \times \delta \mapsto 2^\pi$ such that, for some simulation relation R, it is the case that: $\langle v_0, s_0\rangle \in R$; and for all pairs $\langle v, s\rangle \in R$ and all transitions $d = \langle v, \langle \gamma, \psi, \phi\rangle, v'\rangle$ in $\mathcal{P}$ such that $s \models \gamma$, a plan $\Omega(s, d)$ is defined, realizes transition d, and preserves R from $\langle v, s\rangle$ for d. Essentially, a realization Ω is a function that, given a $\mathcal{D}$-state s and a transition request $\langle v, \langle \gamma, \psi, \phi\rangle, v'\rangle$ whose guard is satisfied in s, outputs a plan π that achieves ϕ while maintaining ψ from s and guarantees that all potential future requests from v' after π's execution can also be fulfilled by plans prescribed by the realization function Ω.

Example 2 (Example 1 cont.) *Consider the p-program of Figure 1.* Can the traveller carry it out? If so, how? *As an example of positive answers, consider Table 1, which describes a possible realization for this p-program. The first column represents the current state of the domain; the second one contains the requested program transition; and the third one represents the plan to be executed from the current domain state in order to realize the requested transition. For simplicity, the second column includes only the source and target state of a program transition, while the corresponding achievement goals are specified in Figure 1. (For the sake of simplicity, maintenance goals and guards are assumed to be true.) Lastly, the third column reports the corresponding plan. Note that, with the realization function in Table 1, transition $\langle v_0, v_1\rangle$ is pursued by the traveller from two different domain states s_0 and s_2, and hence the realization function provides, for this transition, two different plans (π_1 and π_3).*

Planning-based Algorithm

We address the problem of *effectively* constructing *p*-program realizations for deterministic domains by exploiting plan generation techniques for *multiple planning problems* with *preferred end-states* (shortly, PESs) and *tabu end-states* (TESs). Informally, a multiple planning problem is a sequence of planning problems such that the solution plan of each of these problems realizes a program transition. In accordance with the terminology used in (De Giacomo *et al.* 2016), a PES is a desired end state for a plan realizing a planning program transition, while a TES is a forbidden plan end state.

Definition 1 *A multiple planning problem with PESs and TESs over a planning horizon of n p-program transitions is a tuple $\langle A, P, s_0, \{\psi^i\}, \{\phi^i\}, \{S^i_P\}, \{S^i_T\}\rangle$ such that $1 \leq i \leq n$, where A is a set of actions, P is a set of propositions, s_0 is the initial state, $\psi^i \in \Phi(P)$ is a maintenance goal, $\phi^i \in \Phi(P)$ is an achievement goal, $S^i_P \subseteq 2^P$ is a set of PESs, and, finally, $S^i_T \subseteq 2^P$ is a set of TESs.*

State	Transition	Plan
$s_0 =$ { at(A,Pa), at(T,Pa), lev(FL2) }	$\langle v_0, v_1\rangle$	$\pi_1 =$ ⟨ Board(Pa), Fly(Pa,NY,FL2,FL1), Debark(NY) ⟩
$s_1 =$ { at(A,NY), at(T,NY), lev(FL1) }	$\langle v_1, v_0\rangle$	$\pi_2 =$ ⟨ Board(NY), Fly(NY,Pa,FL1,FL0), Refill(FL0) Fly(Pa,Lo,FL2,FL1), Debark(Lo) ⟩
$s_2 =$ { at(A,Lo), at(T,Lo), lev(FL1) }	$\langle v_0, v_1\rangle$	$\pi_3 =$ ⟨ Board(Lo), Fly(Lo,Pa,FL1,FL0), Refill(FL0) Fly(Pa,NY,FL2,FL1), Debark(NY) ⟩

Table 1: An example of realization function for the traveller's planning program.

A multiple plan of length k, $\pi = \langle \pi_1, \ldots, \pi_k\rangle$, is a sequence of k plans $\pi_1, \ldots, \pi_k$. A solution of a multiple planning problem with PESs and TESs is defined as follows.

Definition 2 *Given a multiple planning problem with PESs and TESs over a planning horizon of n transitions, $\Pi = \langle A, P, s_0, \{\psi^i\}, \{\phi^i\}, \{S^i_P\}, \{S^i_T\}\rangle$ with $1 \leq i \leq n$, we say that a multiple plan $\pi = \langle \pi_1, \ldots, \pi_k\rangle$ for some $k \leq n$ is a solution of Π iff the following conditions hold for $j = 1 \ldots k$.*

1. *$Last(\pi_j(s_{j-1})) = s_j \models \phi^j$;*
2. *$s_j \notin S^j_T$;*
3. *π_j maintains ψ^j;*
4. *$k = n$ or s_k is a preferred state in S^k_P.*

Condition (4) means that the execution of multiplan π achieves all the n achievement goals $\{\phi^i \mid 1 \leq i \leq n\}$, or it achieves the k achievement goals $\{\phi^i \mid 1 \leq i \leq k\}$ and ends into a preferred end-state.

Figure 2 shows the pseudo-code of RealizePlanProg$^+$, our algorithm for building *p*-program realizations, which enhances the basic one presented in (De Giacomo *et al.* 2016). Starting from an open configuration (called *open pair* in the algorithm) $\langle s, v\rangle$, where s is a domain state and v is a *p*-program state (initially $s = s_0$ and $v = v_0$), for each transition d outgoing from v such that the guard of d holds in s, RealizePlanProg$^+$ non-deterministically selects a program transition *path* $\langle d_1, \ldots, d_k\rangle$ formed by at most n transitions ($k \leq n$) such that $d = d_1$, constructs a multiple planning problem with PESs and TESs from such a path, and invokes procedure Plan to compute, for some $j \leq k$, a solution multiplan $\langle \pi_1, \ldots, \pi_j\rangle$ of the constructed problem such that every plan π_j realizes transition d_j. For each generated pair $\langle s, v\rangle$ and transition $d = \langle v, \langle \gamma, \psi, \phi\rangle, v'\rangle$ such that $s \models \gamma$, function $\Omega(s, d)$ associates with s plan π_1. Then, the algorithm progresses the states of $\mathcal{D}$ and $\mathcal{P}$ (according to $\pi_1(s)$ and d, respectively), possibly generating a new open pair $\langle s', v'\rangle$ to process similarly.

Algorithm: RealizePlanProg$^+(\mathcal{P}, s_0, n)$

Input: a *p*-program $\mathcal{P} = \langle A, P, V, v_0, \delta \rangle$, an initial state s_0, a planning horizon n;

Output: a realization of $\mathcal{P}$ from s_0 (Function Ω), or `failure`.

1. $\forall s, d \cdot \Omega(s, d) \leftarrow$ `noPlan`;
2. $States(v_0) \leftarrow \{s_0\}$; $\forall v \neq v_0 \cdot States(v) \leftarrow \emptyset$;
3. $\forall v \cdot Tabu(v) \leftarrow \emptyset$;
4. $Open \leftarrow \{\langle s_0, v_0 \rangle\}$;
5. **while** $Open$ is *not* empty **do**
6. **extract** an open pair $\langle s, v \rangle \in Open$;
7. **foreach** $\mathcal{P}$ transition $d = \langle v, \langle \gamma, \psi, \phi \rangle, v' \rangle \in \delta$ **do**
8. **if** $\Omega(s, d) =$ `noPlan` **and** $s \models \gamma$ **then**
9. $w \leftarrow v'$;
10. $k \leftarrow 1$;
11. **while** $k < n$ **do**
12. **add** ψ to Ψ
13. **add** ϕ to Φ
14. **add** $States(w)$ to S_P
15. **add** $Tabu(w)$ to S_T
16. **if** there exists a $\mathcal{P}$ transition outgoing from w
17. **then** select $d = \langle w, \langle \gamma, \psi, \phi \rangle, w' \rangle$;
18. **else break**;
19. $w \leftarrow w'$;
20. $k \leftarrow k + 1$;
21. $\langle \pi_1, \ldots, \pi_j \rangle|_{j \leq k} \leftarrow$ Plan$(A, P, s, \Psi, \Phi, S_P, S_T)$;
22. **if** Plan fails **then break**;
23. **else**
24. $\Omega(s, d) \leftarrow \pi_1$;
25. **if** $Last(\pi_1(s)) \notin States(v')$ **then**
26. **add** $\langle Last(\pi_1(s)), v' \rangle$ **to** $Open$;
27. **add** $Last(\pi_1(s))$ **to** $States(v')$;
28. **add** $\langle s, d \rangle$ **to** $Source(Last(\pi_1(s)), v')$;
29. **if** Plan fails **then**
30. **if** $\langle s, v \rangle = \langle s_0, v_0 \rangle$ **then return** `failure`;
31. **else**
32. **add** s **to** $Tabu(v)$;
33. **remove** s **from** $States(v)$;
34. **foreach** $\langle s'', d'' \rangle \in Source(s, v)$ **do**
35. $\Omega(s'', d'') \leftarrow$ `noPlan`;
36. $Open = Frontier(\Omega, \tau, s_0, v_0)$;
37. **return** Ω.

Figure 2: Algorithm for realizing a planning program $\mathcal{P}$ from state s_0 over a horizon of n *p*-program transitions.

If the algorithm generates an open pair $\langle s, v \rangle$ such that for some transition outgoing from v no realizing plan can be computed from s, backtracking is required, i.e., the plans generating $\langle s, v \rangle$ need to be removed from Ω. The algorithm terminates when no more open pairs are left, or it is the case that no realization can be found, i.e., for at least a transition $d = \langle v_0, \langle \gamma, \psi, \phi \rangle, v \rangle$ outgoing from the initial $\mathcal{P}$-state v_0, and such that γ holds in the initial domain state s_0, there exists no plan π constructed from s_0 such that π maintains ψ, $Last(\pi(s_0)) \models \phi$ and $Last(\pi(s_0))$ is in the set of $\mathcal{D}$-states from which a transition outgoing from v can be realized.

The main difference with the algorithm presented in (De Giacomo *et al.* 2016) is that such an algorithm realizes each *single* program transition $d = \langle v, \langle \gamma, \psi, \phi \rangle, v' \rangle$ *independently from other program transitions*, while the algorithm in Figure 2 realizes d considering *a transition path starting with transition* d. Note that, although the pseudocode in Figure 2 computes from some $\mathcal{D}$-state s a multiplan π realizing a transition path, in this version of the algorithm only the first part π_1 of π, which realizes the first transition $d = \langle v, \langle \gamma, \psi, \phi \rangle, v' \rangle$ of the path, is used to define Ω. The reason why the rest of the plan is discarded is that, while π_1 is generated by guaranteeing that the next $k-1$ transitions on the considered path can be realized from $\langle Last(\pi_1(s)), v' \rangle$, the successive sub-plans in π are guaranteed to be part of a valid realization of the *p*-program with a shorter horizon of program transitions, and hence it is more likely that they compromise the existence of a simulation relation w.r.t. π_1.

Example 3 (Example 2 cont.) *Consider the planning program in Figure 1, and assume $n = 2$. Then, in the first iteration of loop 7–28, the path selected by the algorithm is formed by the p-program transition d_1 from v_0 to v_1 and the transition d_2 from v_1 to v_0. The multiplan computed by* Plan *to solve the multiple planning problem constructed from such a path is $\langle \pi_1, \pi_2 \rangle$ with $\pi_1 = \langle$*Board(Pa)*,* Fly(Pa,NY,FL2,FL1)*,* Debark(NY)*$\rangle$ and $\pi_2 = \langle$*Board(NY)*,* Fly(NY,Lo,FL1,FL0)*,* Debark(Lo)*$\rangle$. If the algorithm progressed the states of $\mathcal{D}$ and $\mathcal{P}$ according to π_1 and d_1 and to π_2 and d_2, then the new open pair $\langle s', v_0 \rangle$ would be $\langle \{$*at(A,Lo)*,* at(T,Lo)*,* lev(FL0)*$\}, v_0 \rangle$. Then, in the next iteration of loop 7–33, backtracking would be required for the open pair $\langle s', v_0 \rangle$, since from s' the airplane cannot be used to move the traveller anymore, as its fuel is over, the airplane is at* Lo*, and the fuel can be recharged only at* Pa*. On the contrary, the first part π_1 of the computed plan can be part of a valid p-program realization, as indicated in Table 1.*

The specification of the function Ω under construction implicitly defines the set of open pairs, also called the *realization frontier*, which in the algorithm is denoted as $Open$. This set is obtained by considering all possible planning program executions, starting from $\langle s_0, v_0 \rangle$, using Ω to realize the transitions, and putting in the set all those pairs $\langle s, v \rangle$ such that for some transition d from v, the guard of d holds in s and $\Omega(s, d)$ is currently undefined. Essentially, this corresponds to a straightforward visit of the *p*-program graph from v_0 and s_0 using the current (partially defined) Ω. The frontier of this visit is the set of pairs $\langle s, v \rangle$, of domain and *p*-program state, such that there is a transition d outgoing from v whose guard holds in s, but for which there is no plan achieving and mainteining the corresponding goal, i.e., $\Omega(s, d)$ is undefined. Such a frontier is denoted by $Frontier(\Omega, \tau, s_0, v_0)$ and defines the open pairs for the current Ω stored in $Open$.

Algorithm RealizePlanProg$^+$ maintains three auxiliary functions $States : V \rightarrow 2^S$, $Tabu : V \rightarrow 2^S$ and $Source : S \times V \rightarrow 2^{S \times \delta}$. Intuitively, $States(v)$ records all domain states reached when $\mathcal{P}$ is in v, for some $\mathcal{P}$ execution, according to the current Ω; $Tabu(v)$ indicates the states of $\mathcal{D}$ that are forbidden when v is reached; and $Source$ associates each open pair $\langle s', v' \rangle$ with those pairs $\langle s, d \rangle$ such that d is a program transition from v to v' and, for $\pi = \Omega(s, d)$, $Last(\pi(s)) = s'$. Essentially, function $Source$ says how an

open pair was generated by the current definition of Ω.

Initially (lines 1–4), Function Ω is completely undefined (through the special value `noPlan`), $States(v) = \emptyset$ for every $v \neq v_0$, $States(v_0) = \{s_0\}$, $Tabu(v) = \emptyset$ for every v, and $Open = \langle s_0, v_0 \rangle$. At each iteration of the external loop (lines 5–36), an arbitrary open pair $\langle s, v \rangle$ is extracted from $Open$ and processed by:

(i) computing, for each transition $d = \langle v, \langle \gamma, \psi, \phi \rangle, v' \rangle$ such that $s \models \gamma$ and $\Omega(s, d) = \texttt{noPlan}$ (i.e., d has not been processed for s yet), a plan π that maintains ψ, achieves ϕ from s with an acceptable end state, i.e., $Last(\pi(s)) \notin Tabu(v')$, and guarantees that the next $k-1$ transitions can be realized from $\langle Last(\pi(s)), v' \rangle$ (lines 7–21);

(ii) appropriately updating Ω, $Open$, and the auxiliary functions (lines 22–36).

When $Open$ becomes empty, the external loop terminates and the algorithm returns Ω (line 37).

Task *(i)* is accomplished by non-deterministically selecting a transition path (lines 16-17), constructing from such a path a set Ψ of maintenance goals, a set Φ of achieving goals, a set S_P of sets of preferred states, and a set S_T of sets of tabu states (lines 11–15), and executing function Plan, which computes a multiplan for the multiple planning problem with PESs and TESs $\langle A, P, s, \Psi, \Phi, S_P, S_T \rangle$.

Intuitively, multiple planning problems are used to improve the chance that there exists a simulation relation from the pair obtained by progressing the states of $\mathcal{D}$ and $\mathcal{P}$ according to first part of the computed multiplan, domain states in $States(v')$ are used as preferred end states to minimize the number of generated open pairs, while the domain states in $Tabu(v')$ are used as tabu end states to prevent next iterations from generating unrealizable open pairs. Details about how to achieve this behavior in Plan are given in the next section.

In our implementation of step 17 of RealizePlanProg$^+$, we select the transition d according to a longest transition path from w that starts with d among the transitions outgoing from w that are not in the transition path constructed so far by loop 11–20. The rationale of this criterion is the following. If two longest transition paths starting from w, p_1 and p_2, are such that p_1 is longer than p_2, then we heuristically estimate that realizing p_1 makes the (sub)plan constructed for realizing the first transition in p_1 more "robust" (deadlock wise) than the (sub)plan for the first transition in p_2. The construction of a longest transition path starting from w and formed by (at most) n transitions is not trivial because the underlying graph may contain cycles. Our algorithm for deriving such a path constrains the computed path to iterate at most once in a cycle (this is the reason why among the transitions outgoing from w we exclude those that are already present in the path under construction). Moreover, in order to deal with cycles, we identify the strongly connected components (SCC) and force the computed path to cross only once every edge connecting vertices in the same SCC. Formally, we select the transition according to the highest value of function $h : V \mapsto \mathbb{N}$ defined as follows.

$$h(v) = \begin{cases} h'(v) & \textit{if } v \textit{ is a vertex of } \mathcal{G} \\ h'(SCC(v)) & \textit{otherwise} \end{cases}$$

where $\mathcal{G}$ is the directed graph obtained by contracting the strongly connected components of the *p*-program transition graph into meta-vertices, $SCC(v)$ is the strongly connected component including v in $\mathcal{G}$, and $h' : V \mapsto \mathbb{N}$ is defined as:

$$h'(v) = \begin{cases} c(v) & \textit{if there is no outgoing edge from } v \textit{ in } \mathcal{G} \\ c(v) + \max\limits_{w \in out(v)} h'(w) & \textit{otherwise} \end{cases}$$

where $out(v) = \{w \mid (v, w) \textit{ is an edge of } \mathcal{G}\}$; $c : V \mapsto \mathbb{N}$ is a function such that $c(v) = n(v)$ if v has no outgoing edge in $\mathcal{G}$, and $c(v) = n(v) + 1$ otherwise; finally, $n : V \mapsto \mathbb{N}$ is a function such that $n(v) = 0$ if v is a simple vertex of $\mathcal{G}$, and $n(v)$ is equal to the number of edges of the *p*-program transition graph contracted in v if v is a meta-vertex of $\mathcal{G}$.

For task *(ii)*, assume that $\langle s, v \rangle$ is an open pair, and d is a program transition from program state v to program state v', whose guard holds in s. If a multiplan $\pi = \langle \pi_1, \ldots, \pi_j \rangle$ from some $j \leq k$ realizing d from s is found, then the algorithm updates $\Omega(s, d)$, $States(v')$ and $Source(s', v')$ as follows: function Ω is updated by setting $\Omega(s, d)$ to π_1; if $s' = Last(\pi_1(s))$ is not already in $States(v')$, the set of open pairs is extended with $\langle s', v' \rangle$; state s' is added to $States(v')$; and $\langle s, d \rangle$ is added to $Source(s', v')$ (lines 24–28). If, for some program transition d outgoing from v such that its guard holds in s, procedure Plan is unable to find a plan achieving/maintaining the goals of d from s and guaranteeing that the next possible $k-1$ transitions can be realized, then open pair $\langle s, v \rangle$ cannot be realized. In the special case $s = s_0$ and $v = v_0$, no realization of $\mathcal{P}$ can be built, and hence RealizePlanProg$^+$ terminates returning failure (lines 29–30). Otherwise ($s \neq s_0$ or $v \neq v_0$), backtracking is performed on Ω (lines 31–36): s is added to $Tabu(v)$; s is removed from $States(v)$, as clearly no longer preferred; for all pairs $\langle s'', v'' \rangle \in Source(s, v)$, $\Omega(s'', d'')$ is set undefined ($\Omega(s'', d'')$ becomes `noPlan`), as the corresponding plans need to be recomputed in order to avoid generating the configuration $\langle s, v \rangle$; and, finally, $Frontier(\Omega, \tau, s_0, v_0)$ defines the new set of open pairs ($Open$).

Compilation Scheme to Classical Planning

In this section, we propose a scheme to transform a multiple planning problem Π with PESs and TESs into a problem Π' with action costs. With such a scheme, if a planner finds a solution plan of Π' with the lowest cost, such a plan can be easily transformed into a solution multiplan of Π ending in one of the PESs of Π.

Definition 3 *A planning problem with action costs is a tuple $\langle A, P, s_0, \phi, c \rangle$, where s_0 is the initial state, $\phi \in \Phi(P)$ is an achievement goal; and $c : A \mapsto \mathbb{R}$ is an action cost function.*

A multiple planning problem with PESs and TESs over a planning horizon of n transitions, $\Pi = \langle A, P, s_0, \{\psi^i\}, \{\phi^i\}, \{S_P^i\}, \{S_T^i\} \rangle$ such that $1 \leq i \leq n$, can be translated into a planning problem with action costs $\Pi' = \langle A', P', s_0, \phi', c \rangle$ such that:

- $P' = P \cup P_C \cup P_T$;
- $A' = \{A_i \mid 1 \leq i \leq n\} \cup A_C \cup A_P \cup A_T$;
- $\phi' = \texttt{completed}(n) \wedge \texttt{check-pref}$
- $c(o) = \begin{cases} 1 & \textit{if } o = \texttt{Ignore-pref}, \\ 0 & \textit{otherwise}; \end{cases}$

where

- $P_C = \{\texttt{started}(i), \texttt{completed}(i), \texttt{check-mode}(i) \mid 1 \leq i \leq n\} \cup \{\texttt{check-pref}\}$;
- $P_T = \{\texttt{not-tabu}(s, i) \mid s \in S^i_T, 1 \leq i \leq n\}$;
- $A_i = \{\langle \mathit{Pre} \cup \{\texttt{started}(i)\} \cup \psi^i, \mathit{Eff}^+, \mathit{Eff}^-\rangle \mid \langle \mathit{Pre}, \mathit{Eff}^+, \mathit{Eff}^-\rangle \in A\}$ with $1 \leq i \leq n$;
- $A_C = \{\texttt{start}(i), \texttt{end}(i) \mid 1 \leq i \leq n\}$ where
  $\texttt{start}(1) = \langle\{\neg\texttt{started}(1), \psi^1\}, \{\texttt{started}(1)\}, \emptyset\rangle$;
  $\texttt{start}(i) = \langle\{\neg\texttt{started}(i), \texttt{completed}(i-1), \psi^i\} \cup \{\texttt{not-tabu}(s, i-1) \mid s \in S^{i-1}_T\}, \{\texttt{started}(i)\}, \{\texttt{check-mode}(i-1)\}\rangle$ with $1 < i \leq n$;
  $\texttt{end}(i) = \langle\{\texttt{started}(i), \neg\texttt{completed}(i), \phi^i\}, \{\texttt{completed}(i), \texttt{check-mode}(i)\}, \{\texttt{started}(i)\}\rangle$ with $1 \leq i \leq n$;
- $A_P = \texttt{Ignore-pref} \cup \{\texttt{Sat-pref}(s, j) \mid 1 \leq j \leq n, \; s \in S^j_P\}$, where Ignore-pref is the action $\langle\{\texttt{check-mode}(n), \texttt{completed}(n)\} \cup \{\texttt{not-tabu}(s, n) \mid s \in S^n_T\}, \{\texttt{check-pref}\}, \emptyset\rangle$, and $\texttt{Sat-pref}(s, i)$ is $\langle\{\texttt{check-mode}(i), \texttt{completed}(i)\} \cup \{p \mid p \in S^i_P\} \cup \{\neg p \mid p \notin S^i_P\} \cup \{\texttt{not-tabu}(s, i) \mid s \in S^i_T\}, \{\texttt{check-pref}, \texttt{completed}(n)\}, \emptyset\rangle$ with $1 \leq i \leq n$;
- $A_T = \{a \mid a \in \texttt{Act-tabu}(s, i) \wedge s \in S^i_T \wedge 1 \leq i \leq n\}$, where $\texttt{Act-tabu}(s, i)$ is the set of actions defined as follows: $\texttt{Act-tabu}(s, i) = \{\langle\{\texttt{completed}(i), \texttt{check-mode}(i), \neg p\}, \{\texttt{not-tabu}(s, i)\}, \emptyset\rangle \mid p \in P \wedge p \in s\} \cup \{\langle\{\texttt{completed}(i), \texttt{check-mode}(i), p\}, \{\texttt{not-tabu}(s, i)\}, \emptyset\rangle \mid p \in P \wedge p \notin s\}$.

Theorem 1 *Let $\Pi = \langle A, P, s_0, \{\psi^i\}, \{\phi^i\}, \{S^i_P\}, \{S^i_T\}\rangle$, with $1 \leq i \leq n$, be a solvable multiple planning problem with PESs and TESs over a planning horizon of n transition, and Π' a planning problem with action costs derived from Π by the translating scheme defined above. Then, (1) there exists a valid plan π' for Π'; and (2) for every plan π' solving Π', the plan obtained by removing the actions in $A_C \cup A_T \cup A_P$ from π' and preconditions $\texttt{started}(i)$ and ψ^i from every action in π' is a valid multiplan for Π.*

Proof. (1) Let $\pi = \langle\pi_1, \ldots, \pi_k\rangle$ be a valid multiplan for Π, and s_k the finale state of π. If $k \leq n$ and $s_k \in S^k_P$, we show that a valid plan π' for Π' is formed by the following sequence of actions. For $i = 1$ to k,

1. $\texttt{start}(i)$,
2. a sequence of actions π'_i obtained by replacing each action in π_i with its corresponding action in A_i,
3. $\texttt{end}(i)$,
4. a sequence of actions formed by one action in $\texttt{Act-tabu}(s, i)$ for each TES $s \in S^i_T$,

plus

5. $\texttt{Sat-pref}(s_k, k)$.

If $k = n$ and $s_k \notin S^k_P$, a valid plan π' is formed by actions (1–4) plus action Ignore-pref.

As for the executability of π', consider that actions in A_C, A_P and A_T do not add or delete any proposition in P, and that the difference between the actions in π_i and π'_i does not concern additive or delete effects of P.

For $1 \leq j \leq k$, $\texttt{start}(j)$ is executable because all its preconditions are satisfied. Specifically, let s_{j-1} be the state when action $\texttt{start}(j)$ is executed. Then,

- precondition $\neg\texttt{started}(j) \in s_{j-1}$ because $\texttt{start}(j)$ is executed at most once;
- ψ_j holds in s_{j-1} because plan π_j maintains ψ_j from the beginning;
- $\{\texttt{not-tabu}(s, j-1) \mid s \in S^{j-1}_T\} \subseteq s_{j-1}$ with $1 < j \leq k$, because for each TES $s \in S^{j-1}_T$ an action in $\texttt{Act-tabu}(s, j-1)$ is executed before $\texttt{start}(j)$.

Similarly, $\texttt{end}(j)$ is executable because it is in π' at most once, and in π' $\texttt{start}(j)$ is executed before $\texttt{end}(j)$. Every action a in π'_j is executable because its corresponding action in π_j is executable, π_j maintains ψ_j, and precondition $\texttt{started}(j)$ of a holds as action $\texttt{start}(j)$ is executed before a.

For $1 \leq j \leq k$ and $s \in S^j_T$, at least one action in $\texttt{Act-tabu}(s, j)$ is executable because it is after $\texttt{end}(j)$, and $\gamma(s_0, \langle\pi_1, \ldots, \pi_j\rangle) \notin S^j_T$ since π is a valid multiplan.

$\texttt{Sat-pref}(s_k, k)$ is executable because it is after action $\texttt{end}(k)$, $s_k \in S^k_P$, and for each TES $s \in S^k_T$ it is after action $\texttt{Act-tabu}(s, k)$. Similarly, if π' contains Ignore-pref, such an action is executable because it is after action $\texttt{end}(n)$ and actions $\{\texttt{Act-tabu}(s, k) \mid s \in S^k_T\}$.

Thereby, all the actions in π' are executable. Moreover, π' achives ϕ' because it contains either $\texttt{end}(k)$ and $\texttt{Sat-pref}(s_k, k)$ or $\texttt{end}(n)$ and Ignore-pref.

(2) Since π' is a valid plan for Π', then it contains either (2.1) $\texttt{end}(k)$ and $\texttt{Sat-pref}(s_k, k)$ for some $k \leq n$ or (2.2) $\texttt{end}(n)$ and Ignore-pref. Consider case (2.1). Then, π' is formed as indicated before by items (1–5). We show that there exists a solution multiplan $\pi = \langle\pi_1, \ldots, \pi_k\rangle$ obtained by substituting the actions in $\{\pi'_j\}_{j=1..k}$ with the corresponding actions in A . All the actions in π are executable because plan π' is executable, and, for each action in π, all its preconditions are also preconditions of the corresponding action in π'. By construction of set of actions $\{\texttt{end}(j) \mid 1 \leq j \leq k\}$, and since the difference between the action in π and π' does not concern additive or delete effects of P, plan π satisfies all the achievement goals $\{\phi(j) \mid 1 \leq j \leq k\}$. For $1 \leq j \leq k$, plan π_j maintains ψ_j because ψ_j is a precondition of action $\texttt{start}(j)$ and any action in π'_j has ψ_j as its precondition. Moreover, since π' is valid, by construction of set of actions $\{\texttt{start}(j) \mid 1 \leq j \leq k\}$, for each TES $s \in S^j_T$ and $1 \leq j \leq k$, plan π' contains at least one action $a \in \texttt{Act-tabu}(s)$ achieving conjunct $\texttt{not-tabu}(s, j-1)$. By construction of A_T and $\texttt{Act-tabu}(s, j-1)$, since all actions in π' are executable and the actions in A_C, A_P and A_T do not add/delete propositions of P, $\gamma(s_0, \langle\pi_1, \ldots, \pi_j\rangle) \notin S^j_T$. Finally, since the last action of π' is $\texttt{Sat-pref}(s_k, k)$, the end state of π is in S^k_P. Hence, all the conditions (1–4) in Definition 2 hold, and π is a solution multiplan.

Similarly, in case (2.2), if the last two actions of π' are $\texttt{end}(n)$ and Ignore-pref, then there exists a solution multiplan $\pi = \langle\pi_1, \ldots, \pi_k\rangle$ with $k = n$. In this case, conditions (1–3) of Definition 2 hold for the same arguments as in the

previous case. Condition (4) holds because, even if the last state of π is not preferred, k is equal to n. $\square$

Theorem 2 *Let Π be a multiple planning problem with PESs and TESs that has a solution plan ending in a PES, and Π' a planning problem with action costs obtained from Π by the translating scheme presented above. Then, (1) there exists a plan π' solving Π' such that $c(\pi') = 0$, and (2) for every plan π' solving Π' such that $c(\pi') = 0$, the plan obtained by removing the actions in $A_C \cup A_T \cup A_P$ from π' and substituting the actions in π' with the corresponding actions in A is a valid multiplan solving Π and ending in a PES.*

Proof. (1) Let π be a valid multiplan for Π ending in a PES of Π. By Theorem 1, there exists a valid plan π' which is formed by the actions (1–5) listed at the beginning of the proof of Theorem 1. Since the cost of every action of π' is zero, $c(\pi') = 0$.

(2) By Theorem 1, the multiplan π obtained from π' by removing the actions in A_C, A_P and A_T and substituting the actions in π' with the corresponding actions in A is valid for Π. Since $c(\pi') = 0$ and π' is valid, π' contains an action $\texttt{Sat-pref}(s_k, k)$ achieving the goal conjunct `check-pref` of ϕ', for some PES $s_k \in S_P^k$ and $k \leq n$. By construction of action set A' and action $\texttt{Sat-pref}(s_k, k)$, $\texttt{Sat-pref}(s_k, k)$ can be executed only as the last action of π'. Moreover, by construction of $\texttt{Sat-pref}(s_k, k)$ and since π' is valid, multiplan π must end in a PES of Π. $\square$

Experimental Results

In our experiments, planning programs are constructed over 3 benchmark domains and with 4 different program structures defined by the *p*-program transition relation δ. We modified domains `Blocksworld`, `Storage`, and `ZenoTravel` to admit dead-end states (Bacchus 2001; Gerevini *et al.* 2009; Long & Fox 2003). Specifically, the actions of `Blocksworld` have been constrained so that, every two moves of any block, the block has to be on the table; if this does not happen the block cannot be moved anymore. For the modified `Storage` and `ZenoTravel` domains, hoists and airplanes consume energy, and can recharge only at certain locations. The considered *p*-program structures are: a single cycle (shortly, 1C), multiple binary cycles in sequence (MC), a random sparse directed graph (RS), and a sequence of vertices plus a vertex linked from any vertex of the sequence (S+1). More formally, these structures are defined as follows.

- 1C$[n]$: $\delta = \{\langle v_i, G_i, v_{((i \bmod n)+1)}\rangle \mid v_i \in V, 1 \leq i \leq n\}$;
- MC$[n]$: $\delta = \{\langle v_i, G_i, v_{i+1}\rangle, \langle v_{i+1}, G_{i+n-1}, v_i\rangle \mid v_i \in V, 1 \leq i < n\}$;
- RS$[n]$: $\delta = \{\langle v_i, G_i, w_i\rangle \mid (v_i, w_i) \in E_{Rand}, 1 \leq i \leq |E_{Rand}| = \lceil n \cdot log_2 n \rceil\}$;
- S+1$[n]$: $\delta = \{\langle v_i, G_i, v_{i+1}\rangle, \langle v_j, G_{j+x-2}, v_x\rangle \mid v_i, v_j \in V, 1 \leq i < x-1, 1 \leq j \leq x-1, x = \lceil \frac{n+3}{2} \rceil\}$;

where V is the set of program states, $n = |V|$, E_{Rand} is a set of $\lceil n \cdot \log_2 n \rceil$ randomly selected pairs of program states, and G_x denotes the x-th set of (randomly generated) achievement goals. Unless differently specified, the sets of achievement goals were obtained by using the existing problem generators. For simplicity, we set all the maintenance goals, and transition guards to true (i.e., we assume there are none of them).

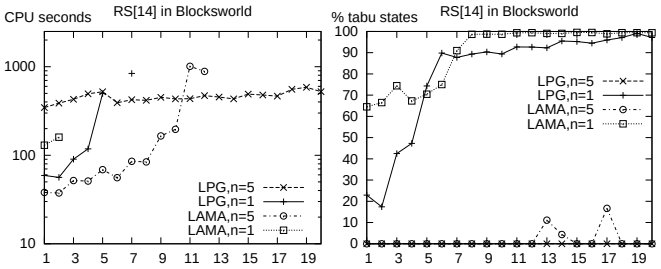

Figure 3: CPU time and percentage of generated tabu states of RealizePlanProg$^+$ using LPG and LAMA with a planning horizon equal to 1 and 5 for planning programs with structure RS$[14]$ (s.t. $|\delta| \approx 50$) over domain `Blocksworld`. The x-axis refers to the number of planning program.

Overall, we constructed 77 planning programs with a randomly generated initial state and $|\delta|$ problem goal sets. Specifically, our benchmark consists of: 20 `Blocksworld` planning programs with a number of blocks ranging from 3 to 12 and program transition relation yielding structure $RS[14]$ ($|\delta| \approx 50$); for each considered domain, 77 planning programs with the same small domain size (the number of domain objects ranges from 8 to 11) and program transition relation yielding structures 1C[5-100], MC[4-51], RS[3-23], and S+1[5-43] ($|\delta|$ ranges from about 5 to 100).

Algorithm RealizePlanProg$^+$ has been tested using two alternative well-known incorporated planners: LAMA (Richter & Westphal 2010), and LPG (Gerevini, Saetti, & Serina 2003). The tests were conducted on an Intel Xeon(tm) 2 GHz machine, with 2 Gbytes of RAM. Unless otherwise indicated, the CPU-time limit used by RealizePlanProg$^+$ to realize planning programs was 20 minutes. The termination of the incorporated planner was forced after 10 minutes or when two different solution plans (with increasing quality) were computed. Note that in this latter case, the second plan necessarily achieves a PES. Moreover, the second plan computed by either LAMA or LPG is an optimal solution (in terms of satisfied PESs). This is because LAMA and LPG minimize the total cost of the plan solving the problem obtained by compiling PESs and TESs away, and, by construction of the compiled problems, at most one action with positive cost can be executed in a valid plan (the cost of every other action is equal to zero).

RealizePlanProg$^+$[X](n) denotes RealizePlanProg$^+$ incorporating planner X and using a planning horizon of n *p*-program transitions. Figure 3 shows the performance of RealizePlanProg$^+$[LAMA] and RealizePlanProg$^+$[LPG] with n equal to 1 and 5 for `Blocksworld` planning programs. Using $n = 1$, the algorithm is the same as the version proposed by De Giacomo *et al.* (2016). The results show that, using $n = 1$, RealizePlanProg$^+$ solves few planning programs because, for most of the `Blocksworld` planning programs, almost every $\mathcal{D}$-state generated by RealizePlanProg$^+$ is a dead-end. Using $n = 5$, the percentage of the generated states that are dead-ends is almost

Domain and structure of p-program	RealizePlanProg$^+$[LPG] 1 Score	#P	2 Score	#P	5 Score	#P	10 Score	#P	25 Score	#P	RealizePlanProg$^+$[LAMA] 1 Score	#P	2 Score	#P	5 Score	#P	10 Score	#P	25 Score	#P
`Blocksworld`																				
1C[5-100]	0.0	0	0.0	0	3.3	4	**19.5**	**20**	13.3	19	0.0	0	0.0	0	2.0	2	**17.4**	**18**	6.9	10
MC[4-51]	0.0	0	1.5	2	13.9	14	**15.7**	**17**	6.0	8	0.0	0	0.0	0	8.0	8	**11.8**	**16**	0.0	0
RS[3-23]	0.0	0	4.6	5	**12.1**	**13**	9.6	11	4.7	6	0.0	0	1.4	2	**18.9**	**19**	11.7	17	0.7	1
`Storage`																				
1C[5-100]	2.9	5	13.7	15	**17.1**	**20**	4.3	7	2.7	5	0.4	1	3.7	5	16.5	18	**17.3**	**20**	**17.3**	**20**
MC[4-51]	0.0	0	2.4	3	**8.2**	**9**	6.3	7	1.7	3	0.0	0	7.0	9	**16.5**	**17**	14.6	**17**	13.1	**17**
RS[3-23]	1.0	1	2.9	3	1.4	2	**4.1**	**5**	2.6	3	0.0	0	4.0	5	11.7	13	**15.9**	**17**	10.1	13
`ZenoTravel`																				
1C[5-100]	1.4	2	0.8	1	4.5	7	14.9	17	**17.0**	**20**	0.9	1	1.0	1	1.9	2	7.6	8	**18.0**	**20**
MC[4-51]	2.2	3	1.6	2	11.6	12	**14.0**	**16**	12.5	15	1.0	2	1.3	2	4.0	4	11.1	12	**14.1**	**17**
RS[3-23]	1.9	3	3.7	4	**7.7**	**9**	6.6	8	5.2	7	0.8	2	4.2	5	15.5	16	**17.9**	**19**	12.3	17
Total	9.5	14	31.1	35	80.0	90	**95.0**	**108**	65.8	86	3.1	6	22.6	29	95.0	99	**125.4**	**144**	92.3	115

Table 2: IPC time score, and number of solved problems of RealizePlanProg$^+$[LPG] and RealizePlanProg$^+$[LAMA] with a planning horizon ranging from 1 to 25 p-program transitions for planning programs with structures 1C[5-100], MC[4-51] and RS[3-23] over domains `Blocksworld`, `Storage` and `ZenoTravel`. Bold numbers indicate the best results.

always equal to zero and RealizePlanProg$^+$ is almost always much faster, and solves many more problems than using $n = 1$. This indicates that for domains with dead-end states, realizing more than one p-program transitions together can be very effective.

The results in Figure 3 raises the question of *how many p-program transitions should be realized "together"*. Table 2 addresses this question by comparing the IPC time score (Helmert, Do, & Refanidis 2010) and the number of solved problems of RealizePlanProg$^+$[LAMA](n) and RealizePlanProg$^+$[LPG](n) with n ranging from 1 to 25. The results show that there is a tradeoff between the planning horizon and the hardness of the multiple planning problems derived by RealizePlanProg$^+$. The higher the planning horizon, the harder the derived multiple planning problems are. Therefore, with a higher planning horizon, it is more likely that LAMA and LPG fail to solve the compiled planning problems. On the other hand, the higher the planning horizon, the more "advised" the computation of a p-program realization is, and hence it is less likely that RealizePlanProg$^+$ generates dead-end states. Overall, according to our experimental results, the best tradeoff is obtained using a planning horizon n equal to 10. For `Storage` and two p-program structures over three, RealizePlanProg$^+$[LPG] performs best when used with $n = 5$, because solving the compiled planning problems derived from these p-program is hard for LPG. On the contrary, for `ZenoTravel` and two p-program structures over three, solving the compiled planning problems derived from these p-program is relatively easy for LAMA, and the best performance is obtained by using $n = 25$.

Another interesting question is *how the sequence of p-program transitions should be selected to be realized together*. To investigate this question, we compare the heuristic based on the longest path, previously presented, with a strategy that randomly selects a p-program transition path of length n. The results of this comparison are in Table 3. This experiment uses planning programs with structure S+1[5-43] where, for almost every $\mathcal{P}$-state v of these planning programs, there are 2 p-program transition paths starting from v of different sizes. Overall, RealizePlanProg$^+$[LAMA] and RealizePlanProg$^+$[LPG] using the longest path heuristic are faster than using a randomly selection of the transitions, and solve many more problems. This means that an accurate selection of the p-program transitions realized together can be very useful.

Domain	Using LPG Random	Using LPG Longest	Using LAMA Random	Using LAMA Longest
`Blocksworld`	3.5 (4)	**17.0 (17)**	1.4 (2)	**12.0 (12)**
`Storage`	16.6 (18)	**18.6 (20)**	18.2 (19)	**18.7 (20)**
`ZenoTravel`	4.4 (6)	**17.0 (17)**	1.7 (2)	**14.0 (14)**
Total	24.5 (28)	**52.6 (54)**	21.2 (23)	**44.7 (46)**

Table 3: IPC time score, and number of solved problems (in parenthesis) of RealizePlanProg$^+$[LPG/LAMA] with the best performing horizon (see Table 2) for planning programs with structure S+1[5-43] in domains `Blocksworld`, `Storage` and `ZenoTravel`.

Conclusions

In this paper, we addressed the problem of effectively constructing planning program realizations over domains with dead-end states. We proposed a significant enhancement of the approach described in (De Giacomo *et al.* 2016), which realizes a program transition while verifying the existance of a solution plan for successive program transitions. Substantially, while the previous basic approach constructs a realization of the planning program by iteratively realizing single program transitions, the enhanced proposed approach constructs the realization by realizing a set of heuristically chosen p-program transitions together.

We provided experimental evidence of the effectiveness of the new technique, and gave (preliminary) experimental results studying a heuristic criterion for selecting the set of program transitions to realize together, and the tradeoff between the size of the set of program transitions realized together and the hardness of the multiple planning problems derived from this set of transitions. An interesting direction for future work is investigating more informative heuristics for selecting the sets of program transitions that it is more useful to realize together.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Bacchus, F. 2001. The AIPS'00 planning competition. *AI Magazine* 22:47–56.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173(5-6):593–618.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 190–197.

De Giacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In *Proceedings of the European Conference on Planning (ECP)*, volume 1809 of *Lecture Notes in Computer Science*, 226–238. Springer.

De Giacomo, G.; Patrizi, F.; Felli, P.; and Sardina, S. 2010. Two-player game structures for generalized planning and agent composition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 297–302.

De Giacomo, G.; Gerevini, A. E.; Patrizi, F.; Saetti, A.; and Sardina, S. 2016. Agent planning programs. *Artificial Intelligence* 231(1):64–106.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Gerevini, A.; Patrizi, F.; and Saetti, A. 2011. An effective approach to realizing planning programs. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 323–326.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)* 20:239–290.

Helmert, M.; Do, M.; and Refanidis, I. 2010. Deterministic part of the 6th International Planning Competition. In *http://icaps-conference.org/ipc2008/deterministic*.

Kabanza, F., and Thiébaux, S. 2005. Search control in planning for temporally extended goals. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 130–139.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysi s. *Journal of Artificial Intelligence Research (JAIR)* 20:1–59.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39:127–177.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence* 175(2):615–647.