Local Search Techniques for Temporal Planning in LPG

Alfonso Gerevini **Ivan Serina** **Alessandro Saetti** **Sergio Spinoni**
Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia
Via Branze 38, I-25123 Brescia, Italy
{gerevini,serina}@ing.unibs.it

Abstract

We present some techniques for planning in temporal domains specified with the recent standard languange PDDL2.1. These techniques are implemented in LPG, a fully-automated system that took part in the third International Planning Competition (Toulouse, 2002) showing excellent performance. The planner is based on a stochastic local search method and on a graph-based representation called "Temporal Action Graphs" (TA-graphs). In this paper we present some new heuristics to guide the search in LPG using this representation. An experimental analysis of the performance of LPG on a large set of test problems used in the competition shows that our techniques can be very effective, and that often our planner outperforms all other fully-automated temporal planners that took part in the contest.

Introduction

Local search is emerging as a powerful method to address domain-independent planning. In particular, two planners that successfully participated in the recent 3rd International Planning Competition (IPC) are based on local search: FF (Hoffmann & Nebel 2001) and LPG. In (Gerevini & Serina 1999; 2002) we presented a first version of LPG using several techniques for local search in the space of *action graphs* (A-graphs), particular subgraphs of the planning graph representation (Blum & Furst 1997). This version handled only STRIPS domains, possibly extended with simple costs associated with the actions. In this paper we present some major improvements that were used in the 3rd IPC to handle domains specified in the recent PDDL2.1 language (Fox & Long 2001) supporting "durative actions" and numerical quantities that were not treated in the first version of LPG.

The general search scheme of our planner is Walkplan, a stochastic local search procedure similar to the well-known Walksat (Selman, Kautz, & Cohen 1994). Two of the most important extensions on which we focus in this paper concern the use of *temporal action graphs* (TA-graphs), instead of simple A-graphs, and some new techniques to guide the local search process. In a TA-graph, action nodes are marked with temporal values estimating the earliest time when the corresponding action terminates. Similarly, a fact node is marked with a temporal value estimating the earliest time when the corresponding fact becomes true. A set of ordering constraints is maintained during search to handle mutually exclusive actions, and to represent the temporal constraints implicit in the "causal" relations between actions in the current plan.

The new heuristics exploit some reachability information to weight the elements (TA-graphs) in the search neighborhood that resolve an inconsistency selected from the current TA-graph. The evaluation of these TA-graphs is based on the estimated number of search steps required to reach a solution (a valid plan), its estimated makespan, and its estimated execution cost. LPG is an incremental planner, in the sense that it produces a sequence of valid plans each of which improves the quality of the previous ones.

In the 3rd IPC our planner showed excellent performance on a large set of problems in terms of both speed to compute the first solution, and quality of the best solution that can be computed by the incremental process.

The 2nd section presents the action and plan representation used in the competition version of LPG; the 3rd section describes its local search neighborhood, some new heuristics for temporal action graphs, and the techniques for computing the reachability and temporal information used in these heuristics; the 4th section gives the results of an experimental analysis using the test problems of the 3rd IPC illustrating the efficiency of our approach for temporal planning; finally, the 5th section gives conclusions and mentions further work.

Action and Plan Representation

Our graph-based representation for temporal plans is an elaboration of *action graphs* (Gerevini & Serina 1999; 2002), particular subgraphs of the planning graph representation. In the following we will assume that the reader is familiar with this well-known representation and with the related terminology. We indicate with $[u]$ the proposition (action) represented by the fact node (action node) u. Given a planning graph $\mathcal{G}$ for a planning problem, without loss of generality, we can assume that the goal nodes of $\mathcal{G}$ in the last level represent the preconditions of the special action $[a_{end}]$, which is the last action in any valid plan, while the fact nodes of the first level represent the effects of the special action $[a_{start}]$, which is the first action in any valid plan.

An *action graph* (A-graph) for $\mathcal{G}$ is a subgraph $\mathcal{A}$ of $\mathcal{G}$ containing a_{end} and such that, if a is an action node of $\mathcal{G}$ in $\mathcal{A}$, then also the fact nodes of $\mathcal{G}$ corresponding to the preconditions and positive effects of $[a]$ are in $\mathcal{A}$, together with the edges connecting them to a. An action graph can contain some *inconsistencies*, i.e., an action with precondition nodes that are not *supported*, or a pair of action nodes involved in a *mutex relation* (or simply in a *mutex*). In general, a precondition node q of level i is supported in an action graph $\mathcal{A}$ of $\mathcal{G}$ if either (i) in $\mathcal{A}$ there is an action node at level $i-1$ representing an action with (positive) effect $[q]$, or (ii) $i=1$ (i.e., $[q]$ is a proposition of the initial state). An action graph without inconsistencies represents a valid plan and is called *solution graph*. A *solution graph* for $\mathcal{G}$ is an action graph $\mathcal{A}_s$ of $\mathcal{G}$ where all precondition nodes of its action nodes are supported, and there is no mutex between its action nodes.

For large planning problems the construction of the planning graph can be computationally very expensive, especially because of the high number of mutex relations. For this reason our planner considers only pairs of actions that are *persistently* mutex (i.e., that hold at every level of the graph). Persistent mutex relations are derived using a dedicated algorithm that is not reported here for lack of space.

The definition of action graph and the notion of supported fact can be made stronger by observing that the effects of an action node can be automatically propagated to the next levels of the graph through the corresponding no-ops, until there is an interfering action *blocking* the propagation (if any), or we reach the last level of the graph. The use of the no-op propagation, which we presented in (Gerevini & Serina 2002), leads to a smaller search space and can be incorporated into the definition of action graph.

An *action graph with propagation* is an action graph $\mathcal{A}$ such that if a is an action node of $\mathcal{A}$ at level l, then, for any positive effect $[e]$ of $[a]$ and any level $l' > l$ of $\mathcal{A}$, the no-op of e at level l' is in $\mathcal{A}$, unless there is another action node at a level l'' ($l \leq l'' < l'$) which is mutex with the no-op. Since in the rest of this paper we consider only action graphs with propagation, we will abbreviate their name simply to action graphs (leaving implicit that they include the no-op propagation).

The first version of LPG (Gerevini & Serina 2002) was based on action graphs where each level may contain an arbitrary number of action nodes, like in the usual definition of planning graph. The newer version of the system that participated in the 3rd IPC uses a restricted class of actions graphs, called *linear action graphs*, combined with some additional data structures supporting a richer plan representation. In particular, the new system can handle actions having temporal durations and preconditions/effects involving numerical quantities specified in PDDL2.1 (Fox & Long 2001). In this paper we focus mainly on planning for temporal domains.

In order to keep the presentation simple, we describe our techniques considering only action preconditions of type "`over all`" (i.e., preconditions that must hold during the whole action execution) and effects of type "`at end`" (i.e., effects that hold at the end of the action execution).[1]

Definition 1 *A* **linear action graph** *(LA-graph) of $\mathcal{G}$ is an A-graph of $\mathcal{G}$ in which each level of actions contains at most one action node representing a domain action and any number of no-ops.*

It is important to note that having only one action in each level of a LA-graph does not prevent the generation of parallel (partially ordered) plans. In fact, from any LA-graph we can easily extract a partially ordered plan where the ordering constraints are (1) those between mutex actions and (2) those implicit in the causal structure of the represented plan. Regarding the first constraints, if a and b are mutex and the level of a precedes the level of b, then $[a]$ is ordered before $[b]$; regarding the second constraints, if a has an effect node that is used (possibly through the no-ops) to support a precondition node of b, then $[a]$ is ordered before $[b]$. These causal relations between actions producing an effect and actions consuming it are similar to the causal links in partial-order planning (e.g., (Penberthy & Weld 1992; Nguyen & Kambhampati 2001)). LPG keeps track of these relationships during search and uses them to derive some heuristic information useful for guiding the search (more details on this in the next section), as well as to extract parallel plans from the solution graph in STRIPS domains.

For temporal domains where actions have durations and plan quality mainly depends on the makespan, rather than on the number of actions or graph levels, the distinction between one action or more actions per level is scarcely relevant. The order of the graph levels should not imply by itself any ordering between actions (e.g., an action at a certain level could terminate before the end of an action at the next level).

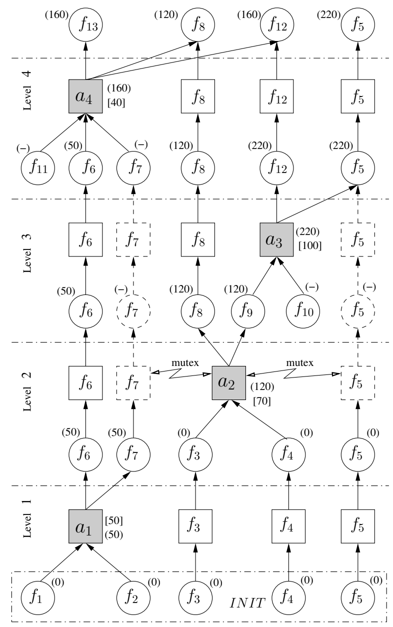
A major advantage of using LA-graphs instead of A-graphs is that the simple structure of LA-graph supports a faster and more accurate computation of the heuristic and reachability information used by the local search algorithm presented in the next section. This is partly related to the fact that in LA-graphs the unsupported preconditions are the only type of inconsistencies that the search process needs to handle explicitly. Moreover, if a level contained mutex actions (possibly because of interacting numerical effects) the resulting state could be indeterminate, and this would make more difficult evaluating our heuristics and reachability information. Finally, having only one action per level allows us to define a larger search neighborhood. A more detailed discussion on LA-graphs versus A-graphs is given in a longer version of this paper (Gerevini & Serina 2003).

For PDDL2.1 domains involving durative actions, our planner represents temporal information by an assignment of real values to the action and fact nodes of the LA-graph, and by a set Ω of *ordering constraints* between action nodes. The value associated with a fact node f represents the (estimated) earliest time at which $[f]$ becomes true, while the value associated with an action node a represents the (estimated) earliest time when the execution of $[a]$ can terminate. These estimates are derived from the duration of the actions in the LA-graph and the ordering constraints between them that are stated in Ω.

Definition 2 *A* **temporal action graph** *(TA-graph) of $\mathcal{G}$ is a triple $\langle \mathcal{A}, \mathcal{T}, \Omega \rangle$ where $\mathcal{A}$ is a linear action graph; $\mathcal{T}$ is an assignment of real values to the fact and action nodes of $\mathcal{A}$; Ω is a set of ordering constraints between action nodes of $\mathcal{A}$.*

The ordering constraints in a TA-graph are of two types: constraints between actions that are implicitly ordered by the causal structure of the plan ($\prec_C$*-constraints*), and constraints imposed by the planner to deal with mutually exclusive actions ($\prec_E$*-constraints*). $a \prec_C b$ belongs to Ω if

[1]The current version of LPG supports all types of preconditions and effects that can be expressed in PDDL2.1.



$$\Omega = \{a_1 \prec_C a_4;\ a_2 \prec_C a_3;\ a_1 \prec_E a_2;\ a_2 \prec_E a_4\}$$

Figure 1: An example of TA-graph. Dashed edges form chains of no-ops that are blocked by mutex actions. Round brackets contain temporal values assigned by $\mathcal{T}$ to the fact nodes (circles) and the action nodes (squares). The Square-nodes marked with facts are no-ops. The numbers in square brackets represent action durations. "(–)" indicates that the corresponding fact node is not supported.

and only if a is used to achieve a precondition node of b in $\mathcal{A}$, while $a \prec_E b$ (or $b \prec_E a$) belongs to Ω only if a and b are mutex in $\mathcal{A}$ ($a \prec_E b$, if the level of a precedes the level of b, $b \prec_E a$ otherwise). In the next section we will discuss how ordering constraints are stated by LPG during the search. Given our assumption on the types of action preconditions and effects in temporal domains, an ordering constraint $a \prec b$ (where "$\prec$" stands for $\prec_C$ or $\prec_E$) states that the end of $[a]$ is before the start of $[b]$.[2] The temporal value assigned by $\mathcal{T}$ to a node x will be denoted with $Time(x)$, and it is derived as follows. If a fact node f is unsupported, then $Time(f)$ is undefined, otherwise it is the minimum over the temporal values assigned to the actions supporting it. If the temporal value of every precondition nodes of an action node a are undefined, and there is no action node with a temporal value that must precede a according to Ω, then $Time(a)$ is set to the duration of a; otherwise $Time(a)$ is the sum of the duration of a and the maximum over the temporal values of its precondition nodes and the temporal values of the actions nodes that must precede a.

Figure 1 gives an example of TA-graph containing four action nodes ($a_{1...4}$) and several fact nodes representing thirteen facts. Since a_1 supports a precondition node of a_4, $a_1 \prec_C a_4$ belongs to Ω (similarly for $a_2 \prec_C a_3$). $a_1 \prec_E a_2$ belongs to Ω because a_1 and a_2 are persistently mutex (similarly for $a_2 \prec_E a_4$). The temporal value assigned to the facts $f_{1...5}$ at the first level is zero, because they belong to the initial state. a_1 has all its preconditions supported at time zero, and hence $Time(a_1)$ is the duration of a_1. Since $a_1 \prec a_2 \in \Omega$, $Time(a_2)$ is given by the sum of the duration of a_2 and the maximum over the temporal values of its precondition nodes (zero) and $Time(a_1)$. $Time(a_3)$ is the sum of its duration and the time assigned to f_9 at level 3, which is the only supported precondition node of a_3. Since f_9 at level 3 is supported only by a_2, and this is the only supported precondition node of a_3, $Time(a_3)$ is the sum of $Time(f_9)$ and the duration of a_3. Since a_2 must precede a_4 (while there is no ordering constraint between a_2 and a_3), $Time(a_4)$ is the maximum over $Time(a_2)$ and the temporal values of its supported precondition nodes, plus the duration of a_4. Finally, note that f_{12} at the last level is supported both by a_4 and a_3. Since $Time(a_3) > Time(a_4)$, we have that $Time(f_{12})$ at this level is equal to $Time(a_4)$.

Definition 3 *A* **temporal solution graph** *for $\mathcal{G}$ is a TA-graph $\langle \mathcal{A}, \mathcal{T}, \Omega \rangle$ such that $\mathcal{A}$ is a solution LA-graph of $\mathcal{G}$, $\mathcal{T}$ is consistent with Ω and the duration of the actions in $\mathcal{A}$, Ω is consistent, and for each pair $\langle a, b \rangle$ of mutex actions in $\mathcal{A}$, either $\Omega \models a \prec b$ or $\Omega \models b \prec a$.*

While obviously the levels in a TA-graph do not correspond to real time values, they represent a topological order for the $\prec_C$-constraints in the TA-graph (i.e., the actions of the TA-graph ordered according to their relative levels form a linear plan satisfying all $\prec_C$-constraints). This topological sort can be a valid total order for the $\prec_E$-constraints of the TA-graph as well, provided that these constraints are appropriately stated during search. Since LPG states $a \prec_E b$ if the level of a precedes the level of b, and $b \prec_E a$ otherwise, it is easy to see that the levels of a TA-graph correspond to a topological order of the actions in the represented plan satisfying every ordering constraint in Ω.

For planning domains requiring to minimize the plan makespan each element of LPG's search space is a TA-graph. For domains where time is irrelevant (like simple STRIPS domains) the search space is formed by LA-graphs.

In accordance with PDDL2.1, our planner handles both static durations and durations depending on the state in which the action is applied. However, in this paper we will not describe the treatment of dynamic durations, and without loss of generality for the techniques presented in the next section, we assume that action durations are static.

Each action of a plan can be associated with a cost that may affect the plan quality. Like action durations, in general these costs could be either static or dynamic, though the current version of LPG handles only static ones. LPG precomputes the action costs using the metric for the plan quality that is specified in the problem description in the PDDL2.1 field "`:metric`".[3]

Local Search in the Space of TA-graphs

In this section we present some search techniques used in the version of our planner that took part in the 3rd IPC. In

[2] Note that in order to handle all types PDDL2.1 preconditions and effects, LPG can use additional types of ordering constraints involving other endpoints of the constrained actions.

[3] For simple STRIPS domains, where there is no metric expression to minimize, the cost of each action is set to one.

order to simplify the notation, instead of using a and $[a]$ to indicate an action node and the action represented by this node respectively, we will use a to indicate both of them (the appropriate interpretation will be clear from the context).

Basic Search Procedure: Walkplan

The general scheme for searching a solution graph (a final state of the search) consists of two main steps. The first step is an initialization of the search in which we construct an initial TA-graph. The second step is a local search process in the space of all TA-graphs, starting from the initial TA-graph. We can generate an initial A-graph in several ways (Gerevini & Serina 1999). In the current version of LPG, the default initialization strategy is the empty action graph (containing only a_{start} and a_{end}).

Each basic search step selects an inconsistency σ in the current TA-graph $\mathcal{A}$, and identifies the *neighborhood* $N(\sigma, \mathcal{A})$ of σ in $\mathcal{A}$, i.e., the set of TA-graphs obtained from $\mathcal{A}$ by applying a graph modification that resolves σ.[4]

The elements of the neighborhood are weighted according to a function estimating their quality, and an element with the best quality is then chosen as the next possible TA-graph (search state). The quality of an TA-graph depends on a number of factors, such as the number of inconsistencies and the estimated number of search steps required to resolve them, the overall cost of the actions in the represented plan and its makespan.

In (Gerevini & Serina 1999; 2002) we proposed three general strategies for guiding the local search in the space of A-graphs. The default strategy that we used in all experimental tests is Walkplan, a method similar to a well-known stochastic local search procedure for solving propositional satisfiability problems (Selman, Kautz, & Cohen 1994; Kautz & Selman 1996). According to Walkplan the best element in the neighborhood is the TA-graph which has the *lowest decrease of quality* with respect to the current TA-graph, i.e., it does not consider possible improvements. This strategy uses a *noise parameter* p. Given a TA-graph $\mathcal{A}$ and an inconsistency σ, if there is a modification for σ that does not decrease the quality of $\mathcal{A}$, then this is performed, and the resulting TA-graph is chosen as the next TA-graph; otherwise, with probability p one of the graphs in $N(\sigma, \mathcal{A})$ is chosen randomly, and with probability $1 - p$ the next TA-graph is chosen according to the minimum value of the evaluation function. If a solution graph is not reached after a certain number of search steps (*max_steps*), the current TA-graph and *max_steps* are reinitialized, and the search is repeated up to a user-defined maximum number of times (*max_restarts*).

In (Gerevini & Serina 2002) we proposed some heuristic functions for evaluation the search neighborhood of A-graphs with action costs, but without considering temporal information. Here we present additional, more powerful heuristic functions for LA-graphs and TA-graphs that were used in the 3rd IPC.

[4]The strategy for selecting the next inconsistency to handle may have an impact on the overall performance. The default strategy that we have used in all experiments presented in the next section prefers inconsistencies appearing at the earliest level of the graph.

Neighborhood and Heuristics for TA-graphs

The search neighborhood for an inconsistency σ in a LA-graphs $\mathcal{A}$ is the set of LA-graphs that can be derived from $\mathcal{A}$ by adding an action node supporting σ, or removing the action with precondition σ (in linear graphs the only type of inconsistencies are unsupported preconditions). An action a supporting σ can be added to $\mathcal{A}$ at any level l preceding the level of σ, and such that the desired effect of a is not blocked before or at the level of σ. The neighborhood for σ contains an action graph for each of these possibilities.

Since at any level of a LA-graph there can be at most one action node (plus any number of no-ops), when we remove an action node from $\mathcal{A}$, the corresponding action level becomes "empty" (it contains only no-ops). When we add an action node to a level l, if l is not empty, then the LA-graph is extended by one level, all action nodes from l are shifted forward by one level, and the new action is inserted at level l.[5] Moreover, when we remove an action node a from the current LA-graph, we can remove also each action node supporting only the preconditions of a. Similarly, we can remove the actions supporting only the preconditions of other removed action, and so on. While this induced pruning is not necessary, an experimental analysis showed that it tends to produce plans of better quality more quickly.

The elements of the neighborhood are evaluated according to an *action evaluation function* E estimating the cost of adding ($E(a)^i$) or removing an action node a ($E(a)^r$). In general, E consists of three weighted terms evaluating three aspects of the quality of the current plan that are affected by the addition/removal of a:

$$\begin{aligned}E(a)^i =& \alpha \cdot Execution\_cost(a)^i + \beta \cdot Temporal\_cost(a)^i + \\ &+ \gamma \cdot Search\_cost(a)^i \\ E(a)^r =& \alpha \cdot Execution\_cost(a)^r + \beta \cdot Temporal\_cost(a)^r + \\ &+ \gamma \cdot Search\_cost(a)^r\end{aligned}$$

The first term of E estimates the increase of the plan execution cost, the second estimates the end time of a, and third estimates the increase of the number of the search steps needed to reach a solution graph. The coefficients of these terms are used to normalize them, and to weight their relative importance (more on this in the section "Incremental Plan Quality").

We first give a brief intuitive illustration of how the terms of E are evaluated by LPG, and then a more detailed, formal description. Suppose we are evaluating the addition of a at level l of the current action graph $\mathcal{A}$. The three terms of E are heuristically estimated by computing a relaxed plan π_r containing a minimal set of actions for achieving (1) the unsupported preconditions of a and (2) the set Σ of preconditions of other actions in the LA-graph that would become unsupported by adding a (because it would block the no-op propagation currently used to support such preconditions). This plan is relaxed in the sense that it does not consider the

[5]Note that the empty levels can be ignored during the extraction of the plan from the (temporal) solution graph. They could also be removed during search, if the graph becomes too large. Finally, if the LA-graph contains adjacent empty levels, and in order to resolve the selected inconsistency a certain action node can be added at any of these levels, then the corresponding neighborhood contains only one of the resultant graphs.

EvalAdd(a)

Input: An action node a that does not belong to the current TA-graph.

Output: A pair formed by a set of actions and a temporal value t.

$INIT_l \leftarrow Supported\_facts(Level(a))$;
$Rplan \leftarrow$ RelaxedPlan$(Pre(a), INIT_l, \emptyset)$;
$t_1 \leftarrow MAX\{0, MAX\{Time(a') \mid \Omega \models a' \prec a\}\}$;
$t_2 \leftarrow MAX\{t_1, End\_time(Rplan)\}$;
$A \leftarrow Aset(Rplan) \cup \{a\}$;
$Rplan \leftarrow$ RelaxedPlan$(Threats(a), INIT_l - Threats(a), A)$;
return $\langle Aset(Rplan), t_2 + Duration(a)\rangle$.

EvalDel(a)

Input: An action node a that belongs to the current TA-graph.

Output: A pair formed by a set of actions and a temporal value t.

$INIT_l \leftarrow Supported\_facts(Level(a))$;
$Rplan \leftarrow$ RelaxedPlan$(Unsup_facts(a), INIT_l, \emptyset)$.
return $Rplan$.

Figure 2: Algorithms for estimating the search, execution and temporal costs for the insertion (EvalAdd) and removal (EvalDel) of an action node a.

delete-effects of the actions. The derivation of π_r takes into account the actions already in the current partial plan (the plan represented by $\mathcal{A}$). In particular, the actions in the current plan are used to define an initial state for the problem of achieving the preconditions of a and those in Σ. The relaxed subplan for the preconditions of a is computed from the state *INIT*$_l$ obtained by applying the actions in $\mathcal{A}$ up to level $l-1$, ordered according to their corresponding levels. Notice that, as we pointed out in the previous section, the levels in a TA-graph correspond to a total order of the actions of the represented partial-order plan that is consistent with the ordering constraints in Ω (though, of course, this is not necessarily the only valid total order). The relaxed subplan for achieving Σ is computed from *INIT*$_l$ modified by the effects of a, and it can reuse actions in the relaxed subplan previously computed for the preconditions of a.

The number of actions in the combined relaxed subplans (π_r) is used to define a heuristic estimate of the additional search cost that would be introduced by the new action a. This estimate takes into account also the number of supported preconditions that would become unsupported by adding the actions in π_r to $\mathcal{A}$. We indicate these subverted preconditions with $Threats(a)$. Using the causal-link notation of partial-order planners (e.g., (Penberthy & Weld 1992)), $Threats(a)$ can be defined as the set $\{f \mid$ no-op(f) and a are mutex; $\exists\, b, c \in \mathcal{A}$ such that $b \xrightarrow{f} c\}$.

$Temporal\_cost(a)$ is an estimation of the earliest time when the new action would terminate given the actions in π_r.[6] $Execution\_cost(a)$ is an estimation of the additional execution cost that would be required to satisfy the preconditions of a, and is derived by summing the cost of each action a' in π_r ($Cost(a')$). The terms of $E(a)^r$ are estimated in a similar way. More formally, the terms of $E(a)^i$ are defined

[6]Note that the makespan of π_r is not a lower bound for $Temporal\_cost(a)$, because the possible parallelization of π_r with the actions already in $\mathcal{A}$ is not considered.

RelaxedPlan($G, INIT_l, A$)

Input: A set of goal facts (G), the set of facts that are true after executing the actions of the current TA-graph up to level l (*INIT*$_l$), a possibly empty set of actions (A);

Output: A set of actions and a real number estimating a minimal set of actions required to achieve G and the earliest time when all facts in G can be achieved, respectively.

$t \leftarrow \underset{g \in G \cap INIT_l}{MAX} Time(g)$;
$G \leftarrow G - INIT_l$; $ACTS \leftarrow A$; $F \leftarrow \bigcup_{a \in ACTS} Add(a)$;
$t \leftarrow MAX\left\{t, \underset{g \in G \cap F}{MAX} T(g)\right\}$;
forall $g \in G$ such that $g \notin F = \bigcup_{a \in ACTS} Add(a)$
  $bestact \leftarrow$ ChooseAction(g);
  $Rplan \leftarrow$ RelaxedPlan$(Pre(bestact), INIT_l, ACTS)$;
  forall $f \in Add(bestact) - F$
    $T(f) \leftarrow End\_time(Rplan) + Duration(bestact)$;
  $ACTS \leftarrow Aset(Rplan) \cup \{bestact\}$;
  $t \leftarrow MAX\{t, End\_time(Rplan) + Duration(bestact)\}$;
return $\langle ACTS, t\rangle$.

Figure 3: Algorithm for computing a relaxed plan achieving a set of action preconditions from the state *INIT*$_l$.

as follows:

$$Execution\_cost(a)^i = \sum_{a' \in\, Aset(\text{EvalAdd}(a))} Cost(a')$$

$$Temporal\_cost(a)^i = End\_time(\text{EvalAdd}(a))$$

$$Search\_cost(a)^i = |Aset(\text{EvalAdd}(a))| + \sum_{a' \in Aset(\text{EvalAdd}(a))} |Threats(a')|$$

Regarding $E(a)^r$ we have

$$Execution\_cost(a)^r = \sum_{a' \in\, Aset(\text{EvalDel}(a))} Cost(a') - Cost(a),$$

while the other two terms are defined as in $E(a)^i$, except that we use EvalDel instead of EvalAdd.

EvalAdd(a) is a function returning two values (see Figure 2): the set of actions in π_r ($Aset$) and an estimation of the earliest time when the new action a would terminate. Similarly for EvalDel(a), which returns a minimal set of actions required to achieve the preconditions that would become unsupported if a were removed from $\mathcal{A}$, together with an estimation of the earliest time when all these preconditions would become supported. The relaxed subplans used in EvalAdd(a) and EvalDel(a) are computed by RelaxedPlan, a recursive algorithm formally described in Figure 3. Given a set G of goal facts and an initial state *INIT*$_l$, RelaxedPlan computes a set of actions forming a relaxed plan (*ACTS*) for achieving G from *INIT*$_l$, and a temporal value (t) estimating the earliest time when all facts in G are achieved. In EvalAdd and EvalDel these two values returned by RelaxedPlan are indicated with $Aset(Rplan)$ and $End\_time(Rplan)$, respectively. $Supported\_facts(l)$ denotes the set of positive facts that are true after executing the actions at levels preceding l (ordered by their level); $Duration(a)$ the duration of a; $Pre(a)$ the precondition nodes of a; $Add(a)$ the (positive) effect nodes of a; $Num\_acts(p, l)$ an estimated minimum number of actions required to reach p from $Supported\_facts(l)$ (if p is not reachable, $Num\_acts(p, l)$

is set to a negative number). The techniques for computing Num_acts and updating $Time$ are described in the next subsection, where we also propose a method for assigning a temporal value to unsupported fact nodes improving the estimation of the earliest end time for an action with unsupported preconditions.

After having computed the state *INIT*$_l$ using $Supported\_facts(l)$, in step 2 EvalAdd uses RelaxedPlan to compute a relaxed subplan ($Rplan$) for achieving the preconditions of the new action a from *INIT*$_l$. Steps 3–4 compute an estimation of the earliest time when a can be executed as the maximum of the end times of all actions preceding a in $\mathcal{A}$ (t_1) and $End\_time(Rplan)$ (t_2). Steps 5–6 compute a relaxed plan for $Threats(a)$ taking into account a and the actions in the first relaxed subplan.

EvalDel is simpler than EvalAdd, because the only new inconsistencies that can be generated by removing a are the precondition nodes supported by a (possibly through the no-op propagation of its effects) that would become unsupported. $Unsup\_facts(a)$ denotes the set of these nodes.

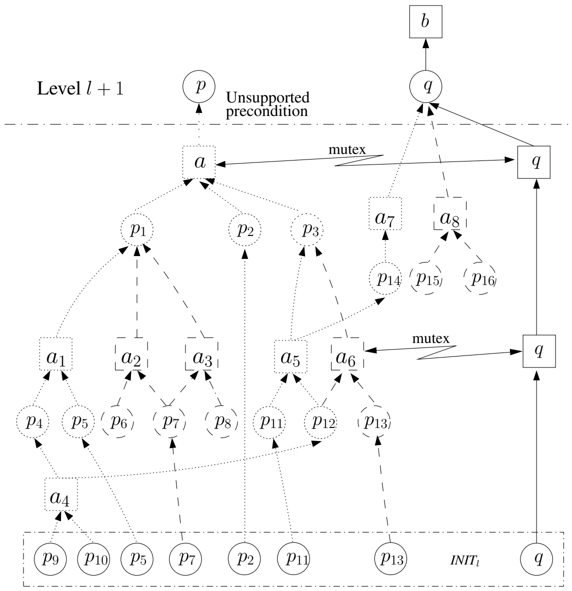
The pair $\langle ACTS, t\rangle$ returned by RelaxedPlan is derived by computing a relaxed plan ($Rplan$) for G starting from a possibly non-empty input set of actions A that can be "reused" to achieve an action precondition or goal of the relaxed problem. A is not empty whenever RelaxedPlan is recursively executed, and when it is used to estimate the minimum number of actions required to achieve $Threats(a)$ (step 6 of EvalAdd(a)). RelaxedPlan constructs $Rplan$ through a backward process where the action chosen to achieve a (sub)goal g ($bestact$) is an action a' such that (1) g is an effect of a'; (2) all preconditions of a' are reachable from *INIT*$_l$; (3) reachability of the preconditions of a' require a minimum number of actions, estimated as the maximum of the heuristic number of actions required to support each precondition p of a' ($Num\_acts(p,l)$); (4) a' subverts the minimum number of supported precondition nodes in $\mathcal{A}$ ($Therats(a')$). More precisely, ChooseAction(g) returns an action satisfying

$$\underset{\{a' \in A_g\}}{ARGMIN}\left\{\underset{p \in Pre(a')-F}{MAX} Num\_acts(p,l) + |Threats(a')|\right\},$$

where $A_g = \{a \in \mathcal{O}\,|\, a \in Add(a)$; $\mathcal{O}$ is the set of all actions; $\forall p \in Pre(a)\ Num\_acts(p) \geq 0\}$, and F is the set of positive effects of the actions currently in *ACTS*.[7]

Steps 1, 3 and 7 estimate the earliest time required to achieve all goals in G. This is recursively defined as the maximum of (a) the times assigned to the facts in G that are already true in the state *INIT*$_l$ (step 1), (b) the estimated earliest time $T(g)$ required to achieve every fact g in G that is an effect of an action currently in *ACTS* (step 3), and (c) the estimated earliest time required to achieve the preconditions of the action chosen to achieve the remaining facts in G (step 10). The T-times of (b) are computed by steps 7–8

[7]The set $\mathcal{O}$ does not contain operator instances with mutually exclusive preconditions. In principle A_g can be empty because g might not be reachable from *INIT*$_l$ (i.e., $bestact = \emptyset$). RelaxedPlan treats this special case by forcing its termination and returning a set of actions including a special action with very high cost, leading E to consider the element of the neighborhood under evaluation a bad possible next search state. For clarity we omit these details from the formal description of the algorithm.



Fact	p_1	p_3	p_4	p_6	p_8	p_{12}	p_{14}	p_{15}	p_{16}
Num_acts	2	2	1	6	2	1	2	2	9

Fact	p_2	p_5	p_7	p_9	p_{10}	p_{11}	p_{13}
$Time$	220	170	300	50	30	170	30

Action	a	a_1	a_4	a_5	a_7
$Duration$	30	70	100	30	90

Figure 4: An example illustrating EvalAdd.

from the relaxed subplan derived to achieve them. Clearly the algorithm terminates, because either every (sub)goal p is reachable from *INIT*$_l$ (i.e., $Num\_acts(p,l) \geq 0$), or at some point $bestact = \emptyset$ holds, forcing immediate termination (see previous footnote).

Figure 4 illustrates EvalAdd and RelaxedPlan with an example. Suppose we are evaluating the addition of a to the current TA-graph $\mathcal{A}$. For each fact that is used in the example, the tables of Figure 4 give the relative Num_acts-value or the temporal value (Num_acts for the unsupported facts, $Time$ for the other nodes). The Num_acts-value for a fact belonging to *INIT*$_l$ is zero. The duration of the actions used in the example are indicated in the corresponding table of Figure 4. Solid nodes represent elements in $\mathcal{A}$, while dotted and dashed nodes represent actions and preconditions considered during the evaluation process. Dotted nodes indicate the actions and the relative preconditions that are selected by RelaxedPlan.

First we describe the derivation of the action set $Aset(Rplan)$ in steps 2 and 6 of EvalAdd(a), and then the derivation of the temporal values t_1 and t_2 in steps 3–4. $Pre(a)$ is $\{p_1, p_2, p_3\}$ but, since $p_2 \in$ *INIT*$_l$, in the first execution of RelaxedPlan only p_1 and p_3 are the goals of the relaxed problem. Suppose that in order to achieve p_1 we can use a_1, a_2 or a_3 (forming the set A_g of step 5). Each of these actions is evaluated by step 5, which assigns a_1 to *bestact*. In the recursive call of RelaxedPlan applied to the preconditions of a_1, p_5 is not considered because it already belongs to *INIT*$_l$. Regarding the other precondition of a_1 (p_4), suppose that a_4 is the only action achieving it. Then this action is chosen to achieve p_4, and since its preconditions belong to INIT_l, they are not evaluated (the new recursive call of RelaxedPlan returns an empty action set).

Regarding the precondition p_3 of a, assume that it can be achieved only by a_5 and a_6. These actions have a common precondition (p_{12}) that is an effect of a_4, an action belonging to *ACTS* (because already selected by RelaxedPlan$(Pre(a_1), \mathit{INIT}_l, \emptyset)$). The other preconditions of these actions belong to INIT_l. Since $|Threats(a_5)| = 0$ and $|Threats(a_6)| = 1$, step 5 of RelaxedPlan selects a_5. Consequently, at the end of the execution of step 2 in EvalAdd(a) we have $Aset(Rplan) = \{a_1, a_4, a_5\}$.

Concerning the execution of RelaxedPlan for $Threats(a)$ = $\{q\}$ in step 6 of EvalAdd(a), suppose that the only actions for achieving q are a_7 and a_8. Since the precondition p_{14} of a_7 is an effect of a_5, which is an action in the input set A (it belongs to the relaxed subplan computed for the preconditions of a), and $Threats(a_7)$ is empty, the best action chosen by RelaxedPlan to support p_{14} is a_7. It follows that the set of actions returned by RelaxedPlan in step 6 of EvalAdd(a) is $\{a_1, a_4, a_5, a_7\}$.

We now describe the derivation of t_2 in EvalAdd(a), which is an estimation of the earliest start time of a. Consider the execution of RelaxedPlan$(Pre(a), \mathit{INIT}_l, \emptyset)$ at step 2. According to the temporal values specified in the table of Figure 4, the value of t at step 1 is $Time(p_2)$ = 220. As illustrated above, RelaxedPlan is recursively executed to evaluate the preconditions of a_1 (the action chosen to achieve p_1) and then of a_4 (the action chosen to achieve p_4). In the evaluation of the preconditions of a_4 (step 1 of RelaxedPlan$(Pre(a_4), \mathit{INIT}_l, \emptyset)$) t is set to 50, i.e., the maximum between $Time(p_9)$ and $Time(p_{10})$, steps 7–8 set $T(p_{12})$ to 50+100 (the duration of a_4), and so RelaxedPlan returns $\langle \emptyset, 50 \rangle$. In the evaluation of the preconditions of a_1 (step 1 of RelaxedPlan$(Pre(a_1), \mathit{INIT}_l, \emptyset)$) t is set to $Time(p_5)$ = 170, and at step 10 it is set to $MAX\{170, 50 + 100\}$. Hence, the recursive execution of RelaxedPlan applied to the preconditions of a_1 returns $\langle \{a_4\}, 170 \rangle$, and at step 10 of RelaxedPlan$(Pre(a), \mathit{INIT}_l, \emptyset)$ t is set to $MAX\{220, 170 + 70\}$ = 240. The recursive execution of RelaxedPlan$(Pre(a_5), \mathit{INIT}_l, \{a_1, a_4\})$ applied to the preconditions of a_5 (the action chosen to achieve p_3) returns $\langle \{a_1, a_4\}, 170 \rangle$. In fact, the only precondition of a_5 that is not true in INIT_l is achieved by an action already in *ACTS* (a_4). Moreover, since $T(p_{12})$ = 150 and $Time(p_{11})$ = 170, the estimated end time of a_5 is 170+30=200. At step 10 of RelaxedPlan$(Pre(a), \mathit{INIT}_l, \emptyset)$ t is then set to $MAX\{240, 200\}$, and the pair assigned to $Rplan$ at step 2 of EvalAdd(a) is $\langle \{a_1, a_4, a_5\}, 240 \rangle$.

Suppose that step 3 of EvalAdd(a) sets t_1 to 230 (i.e., that the highest temporal value assigned to the actions in the TA-graph that must precede a is 230). Step 4 sets t_2 to $MAX\{230, 240\}$, and the execution of RelaxedPlan$(\{q\}, \mathit{INIT}_l - \{q\}, \{a_1, a_4, a_5, a\})$ at step 6 returns $\langle \{a_1, a_4, a_5, a, a_7\}, t_x \rangle$, where t_x is a temporal value that is ignored in the rest of the algorithm, because it does not affect the estimated end time of a. Thus, the output of EvalAdd(a) is $\langle \{a_1, a_4, a_5, a, a_7\}, 240 + 30 \rangle$.

ComputeReachabilityInformation$(I, \mathcal{O})$

Input: the initial state of the planning problem under consideration (I) and all ground instances of the operators in the underlying planning graph ($\mathcal{O}$);
Output: an estimation of the number of actions (Num_acts) and the earliest time ($Time\_fact$) required to achieve each action precondition of the planning problem from I.

1. **forall** facts f
2. **if** $f \in I$ **then**
3. $Num\_acts(f, 1) \leftarrow 0$; $Time\_fact(f, 1) \leftarrow 0$; $Action(f) \leftarrow a_{start}$;
4. **else** $Num\_acts(f, 1) \leftarrow -1$;[8]
5. $F \leftarrow I$; $F_{new} \leftarrow I$; $A \leftarrow \mathcal{O}$;
6. **while** $F_{new} \neq \emptyset$
7. $F \leftarrow F \cup F_{new}$; $F_{new} \leftarrow \emptyset$
8. **while** $A' = \{a \in A \mid Pre(a) \subseteq F\}$ is not empty
9. $a \leftarrow$ an action in A';
10. $ra \leftarrow$ RequiredActions$(I, Pre(a))$;
11. $t \leftarrow \underset{f \in Pre(a)}{MAX} \; Time\_fact(f, 1)$;
12. **forall** $f \in Add(a)$
13. **if** $f \notin F \cup F_{new}$ or $Time\_fact(f, 1) > (t + Duration(a))$ **then**
14. $Time\_fact(f, 1) \leftarrow t + Duration(a)$;
15. **if** $f \notin F \cup F_{new}$ or $Num\_acts(f, 1) > (ra + 1)$ **then**
16. $Num\_acts(f, 1) \leftarrow ra + 1$;
17. $Action(f) \leftarrow a$;
18. $F_{new} \leftarrow F_{new} \cup Add(a) - I$;
19. $A \leftarrow A - \{a\}$;

RequiredActions(I, G)

Input: A set of facts I and a set action preconditions G;
Output: an estimate of the number of the minimum number of actions required to achieve all facts in G from I ($ACTS$).

1. $ACTS \leftarrow \emptyset$;
2. $G \leftarrow G - I$;
3. **while** $G \neq \emptyset$
4. $g \leftarrow$ an element of G;
5. $a \leftarrow Action(g)$;
6. $ACTS \leftarrow ACTS \cup \{a\}$;
7. $G \leftarrow G \cup Pre(a) - I - \bigcup_{b \in ACTS} Add(b)$;
8. **return**$(|ACTS|)$.

Figure 5: Algorithms for computing heuristic information about the reachability of each possible action precondition.

Computing Reachability & Temporal Information

The techniques described in the previous subsection for computing the action evaluation function use heuristic reachability information about the minimum number of actions required to achieve a fact f from INIT_l ($Num\_acts(f, l)$), and earliest times for actions and preconditions. LPG precomputes $Num\_acts(f, l)$ for $l = 1$ and any fact f, i.e., it estimates the minimum number of actions required to achieve f from the initial state I of the planning problem before starting the search. For $l > 1$, $Num\_acts(f, l)$ can be computed only during search because it depends on which are the actions nodes in the current TA-graph (at levels preceding l). Since during search many action nodes can be added and removed, it is impor-

[8] The parser of the planner precomputes all facts of the problem/domain during the phase instantiating the operators.

tant that the computation of $Num\_acts(f, l)$ is fast.

Figure 5 gives ComputeReachabilityInformation, the algorithm used by LPG for computing $Num\_acts(f, 1)$ trying to take account of the tradeoff between quality of the estimation and computational effort to derive it. The same algorithm can be used for (re)computing $Num\_acts(f, l)$ after an action insertion/removal for any $l > 1$ (when $l > 1$, instead of I, in input the algorithm has $Supported\_facts(l)$).[9] In addition to $Num\_acts(f, 1)$, ComputeReachabilityInformation derives heuristic information about the possible earliest time of every fact f reachable from I ($Time\_fact(f, 1)$). LPG can use $Time\_fact(f, 1)$ to assign an initial temporal value to any unsupported fact node representing f, instead of leaving it undefined as indicated above. This can give a more informative estimation of the earliest start time of an action with unsupported preconditions, which is defined as the maximum of the times assigned to its preconditions.

For clarity we first describe the steps of the algorithm considering only Num_acts, and then we comment the computation of $Time\_fact$. In steps 1–4 the algorithm initializes $Num\_acts(f, 1)$ to 0, if $f \in I$, and to -1 otherwise (indicating that f is not reachable). Then in steps 5–19 it iteratively constructs the set F of facts that are reachable from I, starting with $F = I$, and terminating when F cannot be further extended. In this forward process each action is applied at most once, and when its preconditions are contained in the current F. In step 5 the set A of the available actions is initialized to the set of all possible actions, and in step 19 it is reduced after each action application. The internal loop (steps 8–17) applies the actions in A to the current F, possibly deriving a new set of facts F_{new} in step 18. If F_{new} is not empty, F is extended with F_{new} and the internal loop is repeated. Since F monotonically increases and the number of facts is finite, termination is guaranteed. When an action a in A' (the subset of actions currently in A that are applicable to F) is applied, the reachability information for its effects are revised as follows. First we estimate the minimum number ra of actions required to achieve $Pre(a)$ from I using the subroutine RequiredActions (step 10). Then we use ra to possibly update $Num\_acts(f, 1)$ for any effect f of a (steps 12–17). If the application of a leads to a lower estimation for f, i.e., if $ra + 1$ is less than the current value of $Num\_acts(f, 1)$, then $Num\_acts(f, 1)$ is set to $ra + 1$. In addition, a flag indicating the current best action to achieve f ($Action(f)$) is set to a. For any fact f in the initial state, the value of $Action(f)$ is a_{start} (step 3). RequiredActions uses this flag to derive ra through a backward process starting from the input set of action preconditions (G), and ending when $G \subseteq I$. The subroutine incrementally constructs a set of actions (*ACTS*) achieving the facts in G and the preconditions of the actions already selected (using the flag $Action$). Termination of RequiredActions is guaranteed because every element of G is reachable from I.

$Time\_fact(f, 1)$ is computed in a way similar to $Num\_acts(f, 1)$. Step 4 initializes it to 0, for any fact f in the initial state. Then, at every application of an action a in the forward process described above, we estimate the earliest possible time t for applying a as the maximum of the times currently assigned to its preconditions (step 11). For any effect f of a that has not been considered yet (i.e., that is not in F), or that has a temporal value higher than t plus the duration of a, $Time\_fact(f, 1)$ is set to this lower value (because we have found a shorter relaxed plan to achieve f from I).

ComputeReachabilityInformation requires polynomial time in the number of facts and actions in the problem/domain under consideration.

Concerning the ordering constraints in Ω, if during search the planner adds an action node a to $\mathcal{A}$ for supporting a precondition of another action node b, then $a \prec_C b$ is added to Ω. Moreover, for each action c in $\mathcal{A}$ that is mutex with a, if $Level(a) < Level(c)$, then $a \prec_E c$ is added to Ω, otherwise ($Level(c) < Level(a)$) $c \prec_E a$ is added to Ω. If the planner removes a from $\mathcal{A}$, then any ordering constraint involving a is removed from Ω.

The addition/removal of an action node a determines also a possible revision of $Time(x)$ for any fact and action x that is (directly or indirectly) connected to x through the ordering constraints in Ω. Essentially, the algorithm for revising the temporal values assigned to the nodes of $\mathcal{A}$ performs a simple forward propagation starting from the effects of a, and updating level by level the times of the actions (together with the relative precondition and effect nodes) that are constrained by Ω to start after the end of a. When an action node a' is considered for possible temporal revision, $Time(a')$ becomes the maximum temporal values assigned to its precondition nodes plus the duration of a'. The times assigned to the effect nodes of a are revised accordingly. If a' is the only action node supporting an effect f of a, or its temporal value is lower than the value assigned the other action nodes supporting it, then $Time(f)$ is set to $Time(a')$.

Incremental Plan Quality

Our approach can model different criteria of plan quality determined by action costs and action durations. In the current version of LPG the coefficients α, β and γ of the action evaluation function E are used to weight the relative importance of the execution and temporal costs of E, as well as to normalize them with respect to the search cost. Specifically, LPG uses the following function for evaluating the insertion of an action node a (the evaluation function $E(a)^r$ for removing an action node is analogous):

$$E(a)^i = \frac{\mu_E}{max_{ET}} \cdot Execution\_cost(a)^i + \\ + \frac{\mu_T}{max_{ET}} \cdot Temporal\_cost(a)^i + \frac{1}{max_S} \cdot Search\_cost(a)^i,$$

where μ_E and μ_T are non-negative coefficients that weight the relative importance of the execution and temporal costs, respectively. Their values can be set by the user, or they can be automatically derived from the expression defining the plan metrics in the formalization of the problem. The factors $1/max_{ET}$ and $1/max_S$ are used to normalize the terms of E to a value less than or equal to 1. The value of max_{ET} is defined as $\mu_E \cdot max_E + \mu_T \cdot max_T$, where max_E (max_T) is the maximum value of the first (second) term of E over all TA-graphs in the neighborhood, multiplied by the number κ of inconsistencies in the current action graph; max_S

[9] In order to obtain better performance, for $l > 1$ LPG uses an incremental version of ComputeReachabilityInformation updating $Num\_acts(f, l)$ after each action insertion/removal. We omit the details of this version of the algorithm.

Planner	Solved	Attempted	Success ratio
LPG	442	468	94.4%
FF	237	284	83%
Simplanner	91	122	75%
Sapa	80	122	66%
MIPS	331	508	65%
VHPOP	122	224	54%
Stella	50	102	49%
TP4	26	204	13%
TPSYS	14	120	12%
SemSyn	11	144	8%

Table 1: Number of problems attempted and solved by the planners that took part in the 3rd IPC.

is defined as the maximum value of $Search\_cost$ over all possible action insertions/removals that eliminate the inconsistency under consideration.[10]

Without this normalization the first two terms of E could be much higher than the value of the third term. This would guide the search towards good quality plans without paying sufficient attention to their validity.

Our planner can produce a succession of valid plans where each plan is an improvement of the previous ones in terms of its quality. The first plan generated is used to initialize a new search for a second plan of better quality, and so on. This is a process that incrementally improves the quality of the plans, and that can be stopped at any time to give the best plan computed so far. Each time we start a new search, some inconsistencies are forced in the TA-graph representing the previous plan, and the resultant TA-graph is used to initialize the search. Similarly, during search some random inconsistencies are forced in the current when a valid plan that does not improve the plan of the previous search is reached. For lack of space we omit the details of this process.

Experimental Results

In this section we present some experimental results illustrating the efficiency of LPG using the test problems of the 3rd IPC. These problems belong to several domains (including `Depots`, `DriveLog`, `Rovers`, `Satellite` and `Zenotravel`), and each domain has some variants containing different features of PDDL2.1. These variants are named "Strips", "SimpleTime", "Time", "Complex", "Numeric" and "HardNumeric", which are all handled by our planner. For a description of these domains and of the relative variants the reader may see the official web site of the 3rd IPC (`www.dur.ac.uk/d.p.long/competition.html`).

All tests were conducted on the official machine of the competition, an AMD Athlon(tm) MP 1800+ (1500Mhz) with 1 Gbytes of RAM. The results for LPG correspond to median values over five runs of LPG for each problem considered. The CPU-time limit for each run was 5 minutes, after which termination was forced. Notice that the results that we present here are not exactly the same as the official results of the competition, where for lack of time we were not able to run our system a sufficient number of times to obtain meaningful statistical data. However, in general the new results are very similar to those of the competition, with some considerable improvement in `Satellite` Complex and in the `Rovers` domains, where many problems could not be solved due to a bug in the parser of the planner that was fixed after the competition.

[10]The role of κ is to decrease the importance of the first two optimization terms when the current plan contains many inconsistencies, and to increase it when the search approaches a valid plan.

Domain	Problems solved	LPG-s better	LPG-q better	LPG-s worse	LPG-q worse
Simple-time					
`Depots`	21 (11)	18 (81.8%)	19 (86.4%)	3 (13.6%)	1 (4.5%)
`DriverLog`	18 (16)	15 (75%)	17 (85%)	3 (15%)	1 (5%)
`Rovers`	20 (10)	17 (85%)	20 (100%)	1 (5%)	0 (0%)
`Satellite`	20 (19)	18 (90%)	20 (100%)	1 (5%)	0 (0%)
`ZenoTravel`	19 (16)	18 (90%)	17 (85%)	1 (5%)	2 (10%)
`Total`	96(70.6)%	83.4%	91.2%	8.8%	3.9%
Time					
`Depots`	20 (11)	14 (63.6%)	17 (77.3%)	6 (27.3%)	2 (9.1%)
`DriverLog`	18 (16)	17 (85%)	17 (85%)	0 (0%)	1 (5%)
`Rovers`	20 (12)	18 (90%)	18 (90%)	2 (10%)	2 (10%)
`Satellite`	20 (20)	19 (95%)	20 (100%)	1 (5%)	0 (0%)
`ZenoTravel`	19 (20)	15 (75%)	11 (55%)	5 (25%)	9 (45%)
`Total`	95(77.5)%	81.4%	81.4%	13.7%	13.7%
Complex					
`Satellite`	20 (17)	19 (95%)	19 (95%)	1 (5%)	1 (5%)
`Total`	96(75)%	83.9%	87.1%	10.7%	8.5%

Table 2: Summary of the comparison of LPG and the SuperPlanner in terms of: number of problems solved by LPG and the SuperPlanner (in brackets); problems in which LPG-speed (LPG-s) is faster/slower (3rd/5th columns); problems in which LPG-quality (LPG-q) computes better/worse solutions (4th/6th columns).

Overall, the number of problems attempted in the new tests by our planner was 468 (over a total of 508 problems), and the success ratio was 94.4% (the problems attempted by LPG in the competition were 372 and the success ratio 87%). Figure 1 gives these data for every fully-automated planner that took part in the competition. The success ratio of LPG is by far the highest one over all competing planners.

The 40 problems that were not attempted by our planner are the 20 problems in `Settlers Numeric` and the 20 problems in `Satellite` Hardnumeric. The first domain contains operators with universally quantified effects, which are not handled in the current version of LPG; the plan metrics of the problems in the second domain require to maximize a certain expression, which is another feature of PDDL2.1 that currently LPG does not handle properly (in principle, many of these problems could be solved by the empty plan, by we do not consider this an interesting valid solution).

We ran LPG with the same default settings for every problem attempted (maximum numbers of search steps and restarts for each run, and inconsistency selection strategy). The parameters μ_E and μ_T of the action evaluation function were automatically set using the (linear) plan metrics specified in the problem formalizations.

In order to derive some general results about the performance of our planner with respect to all the other planners of the competition, we have compared LPG with the best result over all the other planners. We will indicate these results as if they were produced by an hypothetical "SuperPlanner" (note, however, that such a planner does not exist). Since our main focus in this paper is temporal planning, it is interesting to compare LPG and the SuperPlanner in the temporal variants of the competition domains. The results of this comparison are given in Figure 6 contains detailed results from this comparison on three domains, while Table 2 contains summary results for the SimpleTime, Time and Com-

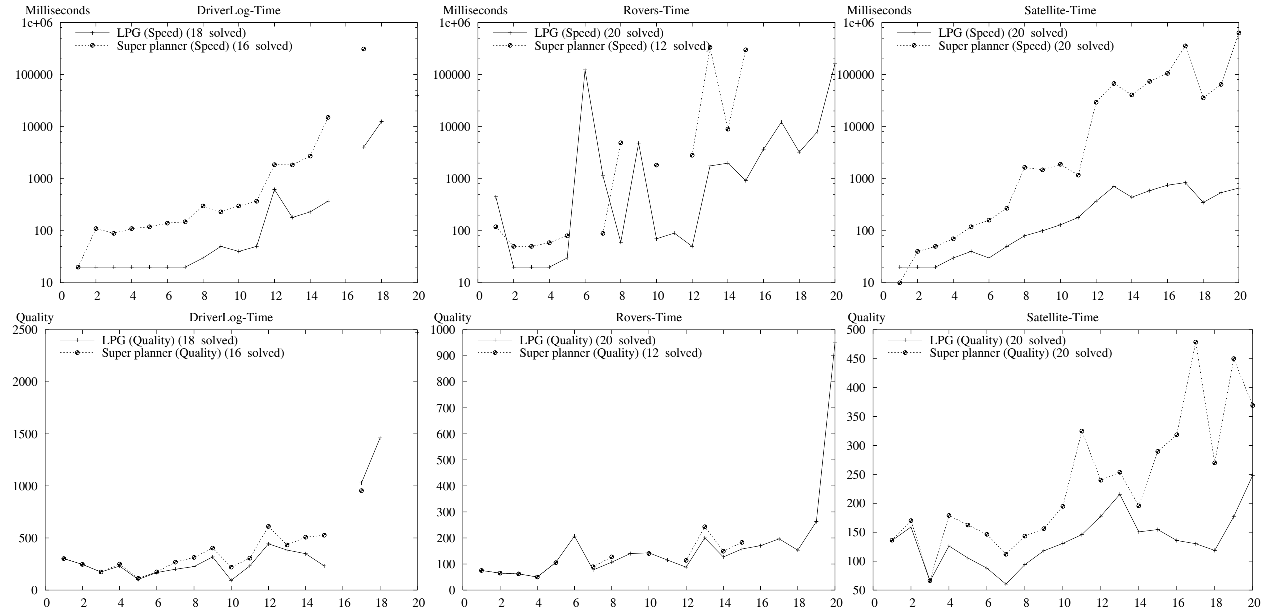


Figure 6: Performance of LPG-speed (left plots) and LPG-quality (right plots) compared with the SuperPlanner in `DriverLog`, `Rovers` and `Satellite` Time. On the x-axis we have the problem names simplifies with numbers. On the y-axis, we have CPU-time (log scale) for the plots of LPG-speed, or the quality of the plans measured using the metric specified in problem formalizations for the plots of LPG-quality.

plex variants.[11] The performance of LPG was tested in terms of both CPU-time required to find a solution (LPG-speed) and quality of the best plan computed, using at most 5 minutes of CPU-time (LPG-quality). LPG-speed is usually faster than the SuperPlanner, and it always solves a larger number of problems, except in `ZenoTravel`, where our planner solves one problem less than the SuperPlanner. Overall, the percentage of the problems solved by LPG-speed is 96%, while those solved by the SuperPlanner is 75%. The percentage of the problems in which our planner is faster is 83.9%, the pecentage of the problems in which it is slower is 10.7%.

Concerning LPG-quality, generally in these domains the quality of the best plans produced by our planner is similar to the quality of the plans generated by the SuperPlanner, with some significant differences in `ZenoTravel`, where in a few problems the SuperPlanner performs better, and in `Satellite`, where our planner performs always better. Overall, the percentage of the problems in which our planner produced a solution of better quality is 87.1%, while for the SuperPlanner this percentage is only 8.5%.

Conclusions

We have presented some new techniques for temporal planning that are implemented in LPG, a planner that was awarded for "distinguished performance of the first order". at the last planning competition. Although we limited our presentation to preconditions of type `over all` and effects of type `at end`, our planner can handle all types of preconditions and effects that can be specified using PDDL2.1.

Further techniques implemented in LPG concern the restriction of the search neighborhood when it contains many elements, and their evaluation slow down the search excessively; the choice of the inconsistency to handle at each search step; the treatment of numerical quantities in the action preconditions and effects.

Current and future work includes further experiments to test other local search schemes and types of graph modifications that we presented in (Gerevini & Serina 1999; 2002), as well as to evaluate the relative impact on performance of the various techniques used by our planner.

[11] Complete results for all domains and planners are available on-line at `http://prometeo.ing.unibs.it/lpg/test-results`

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. http://www.dur.ac.uk/d.p.long/competition.html.

Gerevini, A., and Serina, I. 1999. Fast planning through greedy action graphs. In *Proc. of AAAI-99*.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs with action costs. In *Proc. of AIPS-02*.

Gerevini, A., and Serina, I. 2003. Planning through Stochastic Local Search and Temporal Action Graphs. In *JAIR* (to appear).

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. *Proc. of AAAI-96*.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc. of IJCAI-01*.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. *Proc. of KR'92*.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proc. of AAAI-94*.