Integrating Planning and Temporal Reasoning for Domains with Durations and Time Windows

Alfonso Gerevini[+] **Alessandro Saetti**[+] **Ivan Serina**[*]
[+] Dip. Elettronica per l'Automazione, Università di Brescia, via Branze 38, 25123 Brescia, Italy
[*] Dept. of Computer and Information Systems, University of Strathclyde, Glasgow, UK
E-mail: {gerevini,saetti}@ing.unibs.it ivan.serina@cis.strath.ac.uk

Abstract

The treatment of exogenous events in planning is practically important in many domains. In this paper we focus on planning with exogenous events that happen at known times, and affect the plan actions by imposing that the execution of certain plan actions must be during some time windows. When actions have durations, handling such constraints adds an extra difficulty to planning, which we address by integrating temporal reasoning into planning. We propose a new approach to planning in domains with durations and time windows, combining graph-based planning and disjunctive constraint-based temporal reasoning. Our techniques are implemented in a planner that took part in the 4th International Planning Competition showing very good performance in many benchmark problems.

1 Introduction

In many real-world planning domains, the execution of certain actions can only occur during some predefined time windows where one or more necessary conditions hold. For instance, we can refuel a car at a gas station only during specific period(s) of the day (when the gas station is open). The truth of these conditions is determined by some exogenous events that happen at known times, and that cannot be influenced by the actions available to the planning agent (e.g., the closing of the fuel station).

Several frameworks supporting durations and time windows have been proposed (e.g., [Vere, 1983; Muscettola, 1994; Laborie & Ghallab, 1995; Schwartz & Pollack, 2004]). However, most of them are domain-dependent systems or are not fast enough on large-scale problems. In this paper, we propose a new approach to planning with these temporal features, that combines graph-based planning and constraint-based temporal reasoning.

The last two versions of the language of the International planning competition, PDDL2.1 and PDDL2.2, support planning with action durations and deterministic exogenous events [Fox & Long, 2003; 2004; Edelkamp & Hoffmann, 2004]. In particular, in PDDL2.2, deterministic exogenous events can be represented by *timed initial literals*, one of the new PDDL features on which the 2004 competition focused. Timed initial literals are stated in the description of the initial state of the planning problem through assertions of the form "`(at t L)`", where `t` is a real number, and `L` is a ground literal whose predicate does not appear in the effects of any domain action. The obvious meaning of `(at t L)` is that `L` is true from time `t`. A set of these assertions involving the same ground predicate defines a sequence of disjoint time windows over which the timed predicate holds. An example in the known benchmark domain " Zenotravel" is

```
(at 8 (open-fuelstation city1))
(at 12 (not (open-fuelstation city1)))
(at 15 (open-fuelstation city1))
(at 20 (not (open-fuelstation city1)))
```

These assertions define two time windows over which `(open-fuelstation city1)` is true. A timed initial literal is relevant to the planning process when it is a precondition of a domain action, which we call a *timed precondition* of the action. Each timed precondition of an action can be seen as a temporal scheduling constraint for the action, defining the feasible time window(s) when the action can be executed.

When actions in a plan have durations and timed preconditions, finding a valid plan is a complex task that requires integrating planning and reasoning about time, to check whether the execution of the planned actions can satisfy their scheduling constraints. If an action in the plan cannot be scheduled, the plan is not valid, and it must be revised.

The main contributions and organization of this work are: (i) a new representation of temporal plans with action durations and timed preconditions, integrating disjunctive constraint-based temporal reasoning into a recent graph-based approach to planning (Section 2); (ii) a polynomial method for solving the disjunctive temporal reasoning problems that arise in our context (Section 2); (iii) some new local search heuristics to guide the planning process using our representation (Section 3); (iv) an experimental analysis evaluating an implementation of our approach, showing good performance with respect to other recent domain-independent temporal planners (Section 4).

2 Temporally Disjunctive Action Graph

In our approach, we represent a (partial) plan for a domain with timed initial literals through an extension of the linear action graph representation [Gerevini, et al., 2003], which we call *Temporally-Disjunctive Action Graph* (TDA-graph).

2.1 Background: Linear Action Graph

A linear action graph (LA-graph) $\mathcal{A}$ for a planning problem Π is a directed acyclic leveled graph alternating a *fact level*, and an *action level*. Fact levels contain *fact nodes*, each of which is labeled by a ground predicate of Π. Each fact node

f at a level l is associated with a *no-op* action node at level l representing a dummy action having the predicate of f as its only precondition and effect. Each action level contains one action node labeled by the name of a domain action that it represents, and the no-op nodes corresponding to that level.

An action node labeled a at a level l is connected by incoming edges from the fact nodes at level l representing the preconditions of a (*precondition nodes*), and by outgoing edges to the fact nodes at level $l+1$ representing the effects of a (*effect nodes*). The initial level contains the special action node a_{start}, and the last level the special action node a_{end}. The effect nodes of a_{start} represent the positive facts of the initial state of Π, and the precondition nodes of a_{end} the goals of Π.

A pair of action nodes (possibly no-op nodes) can be constrained by a *persistent mutex relation*, i.e., a mutually exclusive relation holding at every level of the graph, imposing that the involved actions can never occur in parallel in a valid plan. Such relations can be efficiently precomputed using an algorithm given in [Gerevini, et al., 2003].

An LA-graph $\mathcal{A}$ also contains a set of *ordering constraints* between actions in the (partial) plan represented by the graph. These constraints are (i) constraints imposed during search to deal with mutually exclusive actions: if an action a at level l of $\mathcal{A}$ is mutex with an action b at a level after l, then a is constrained to finish before the start of b; (ii) constraints between actions implied by the causal structure of the plan: if an action a is used to achieve a precondition of an action b, then a is constrained to finish before the start of b.

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, or the last level of the graph has been reached.

2.2 Augmenting the LA-graph with Disjunctive Temporal Constraints

Let p be a timed precondition over a set $W(p)$ of time windows. In the following, x^- and x^+ indicate the start time and end time of x, respectively, where x is either a time window or an action. We will describe our techniques focusing on action preconditions that must hold during the whole execution of the action (except at the end of the action, as for PDDL2.1 "over all" conditions), and on operator effects that hold at the end of the action execution.[1]

In order to represent plans where actions have durations and time windows for their possible scheduling, we augment the ordering constraints of an LA graph with (i) action *duration constraints* and (ii) action *scheduling constraints*. Duration constraints have form $a^+ - a^- = Dur(a)$, where $Dur(a)$ denotes the duration of an action a.[2] Duration constraints are supported by the representation presented in [Gerevini, et al., 2003], while the representation and treatment of scheduling constraints are a major contribution of this work.

Let π be the plan represented by an LA-graph $\mathcal{A}$. It is easy to see that the set $\mathcal{C}$ of the ordering constraints in $\mathcal{A}$, extended with the duration constraints of the actions in π, can

[1] Our methods and planner support all the types of operator condition and effect that can be specified in PDDL 2.1 and 2.2.

[2] The duration of a_{start} and a_{end} is 0, $a^-_{start} = a^+_{start}$ and $a^-_{end} = a^+_{end}$.

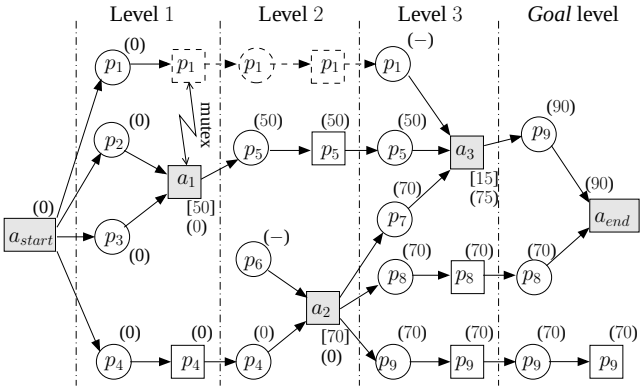

Figure 1: An example of LA-graph with nodes labeled by $\mathcal{T}$-values. Square nodes are action nodes; circle nodes are fact nodes. Dashed edges form chains of no-ops blocked by mutex actions. The $\mathcal{T}$-values are the numbers inside round brackets. The action durations are the numbers inside square brackets. Unsupported precondition nodes are labeled "(–)".

be encoded into a *Simple Temporal Problem* (STP) [Dechter, et al., 1991], i.e., a set of constraints of form $y - x \leq t$, where y and x are point variables and t is a real number. For instance, if $a_i \in \pi$ is used to support a precondition node of a_j, then $a_i^+ - a_j^- \leq 0$ is in $\mathcal{C}$; if a_i and a_j are two mutex actions in π and a_i is ordered before a_j, then $a_i^+ - a_j^- \leq 0$ is in $\mathcal{C}$. Moreover, for every action $a \in \pi$, the following STP-constraints are in $\mathcal{C}$:

$$a^+ - a^- \leq Dur(a), \;\; a^- - a^+ \leq -Dur(a).$$

A scheduling constraint imposes that the execution of an action must occur during the time windows associated with a timed precondition of the action. Syntactically, it is a disjunctive constraint $c_1 \vee \cdots \vee c_n$, where c_i is of the form

$$(y_i^1 - x_i^1 \leq k_i^1) \wedge (y_i^2 - x_i^2 \leq k_i^2),$$

$y_i^1, x_i^1, y_i^2, x_i^2$ are action start times or action end times, and $k_i^1, k_i^2 \in \mathbb{R}$. For every action $a \in \pi$ with a timed precondition p, the following disjunctive constraint is added to $\mathcal{C}$:

$$\bigvee_{w \in W(p)} \big((a^+_{start} - a^- \leq -w^-) \wedge (a^+ - a^+_{start} \leq w^+) \big).$$

Definition 1 *A* **temporally disjunctive action graph** (TDA-graph) *is a 4-tuple $\langle \mathcal{A}, \mathcal{T}, \mathcal{P}, \mathcal{C} \rangle$ where*

- *$\mathcal{A}$ is a linear action graph;*
- *$\mathcal{T}$ is an assignment of real values to the nodes of $\mathcal{A}$;*
- *$\mathcal{P}$ is the set of time point variables corresponding to the start times and end times of the actions labeling the action nodes of $\mathcal{A}$;*
- *$\mathcal{C}$ is a set of ordering constraints, duration constraints and scheduling constraints involving variables in $\mathcal{P}$.*

A TDA-graph $\langle \mathcal{A}, \mathcal{T}, \mathcal{P}, \mathcal{C} \rangle$ represents the (partial) plan formed by the actions labeling the action nodes of $\mathcal{A}$ with start times assigned by $\mathcal{T}$. Figure 1 gives the LA-graph and $\mathcal{T}$-values of a simple TDA-graph. The ordering constraints and duration constraints in $\mathcal{C}$ are:[3]

[3] For brevity, in our examples we omit the constraints $a^+_{start} - a_i^- \leq 0$ and $a_i^+ - a^-_{end} \leq 0$, for each action a_i.

$a_1^+ - a_3^- \leq 0,\ a_2^+ - a_3^- \leq 0,$
$a_1^+ - a_1^- = 50,\ a_2^+ - a_2^- = 70,\ a_3^+ - a_3^- = 15.$

Assuming that p is a timed precondition of a_3 with windows $[25, 50)$ and $[75, 100)$, the only scheduling constraint in $\mathcal{C}$ is:

$(a_{start}^+ - a_3^- \leq -25 \wedge a_3^+ - a_{start}^+ \leq 50) \vee$
$(a_{start}^+ - a_3^- \leq -75 \wedge a_3^+ - a_{start}^+ \leq 100).$

The pair $\langle \mathcal{P}, \mathcal{C} \rangle$ defines a *Disjunctive Temporal Problem* $\mathcal{D}$ (DTP) [Stergiou & Koubarakis, 2000; Tsamardinos & Pollack, 2003].[4] Let $\mathcal{D}_s$ be the set of scheduling constraints in $\mathcal{D}$. We have that $\mathcal{D}$ represents a set Θ of STPs, each of which consists of the constraints in $\mathcal{D} - \mathcal{D}_s$ and one disjunct (pair of STP-constraints) for each disjunction in a *subset* of $\mathcal{D}_s$. We call a consistent STP in Θ an *induced STP* of $\mathcal{D}$. If an induced STP contains a disjunct for every disjunction in $\mathcal{D}_s$, we say that such a (consistent) STP is a *complete induced STP* of $\mathcal{D}$.

An STP is *consistent* iff it has a solution. A *solution* of an STP is an assignment of real values to the variables of the STP that is consistent with every constraint in the STP. Given an induced STP, we can compute in $O(n \cdot c)$ time a solution where each variable has the shortest possible distance from a_{start}^+ [Dechter, et al., 1991; Gerevini & Cristani, 1997], for n variables and c constraints. We call such a solution an *optimal solution* for the induced STP under consideration.

The values assigned by $\mathcal{T}$ to the action nodes of $\mathcal{A}$ are the action start times corresponding to an optimal solution of an induced STP. We call these start times a *schedule* of the actions in $\mathcal{A}$. The $\mathcal{T}$ value labeling a fact node f of $\mathcal{A}$ is the earliest time $t = \mathcal{T}_a + Dur(a)$ such that a supports f in $\mathcal{A}$, and a starts at $\mathcal{T}_a$.

If the induced STP from which we derive a schedule is incomplete, $\mathcal{T}$ may violate the scheduling constraint of some action nodes, that we say are *unscheduled* in $\mathcal{T}$.

The following definition gives a notion of optimality over the complete induced STPs of a DTP that will be used in the next section.

Definition 2 *Given a DTP $\mathcal{D}$ with a point variable p, a complete induced STP of $\mathcal{D}$ is an* **optimal induced STP** *of $\mathcal{D}$ for p, iff it has a solution assigning to p a value that is less than or equal to the value assigned to p by every solution of every other complete induced STP of $\mathcal{D}$.*

An *optimal schedule* is an optimal solution of an optimal induced STP for a_{end}^-. Note that an optimal solution minimizes the makespan of the represented (possibly partial) plan.

2.3 Solving the DTP of a TDA-graph

In general, computing a complete induced STP of a DTP (if it exists) is an NP-hard problem that can be solved by a backtracking algorithm [Stergiou & Koubarakis, 2000; Tsamardinos & Pollack, 2003]. However, given the particular structure of the temporal constraints forming a TDA-graph, we show that this task can be accomplished in polynomial time with a backtrack-free algorithm. Moreover, the algorithm computes an optimal induced STP for a_{end}^-.

[4] The disjunctive constraints in $\mathcal{C}$ are not exactly in DTP-form, i.e., a disjunction $c_1 \vee \cdots \vee c_n$, where c_i is of form $y_i - x_i \leq k_i$, x_i and y_i are time points, and k_i is a real number. However, it is easy to see that every disjunctive constraint in $\mathcal{C}$ can be translated into an equivalent conjunction of constraints in exact DTP-form. We use our more compact notation for clarity and efficiency reasons.

Solve-DTP(X, S)

1. **if** $X = \emptyset$ **then stop** and **return** S;
2. $x \leftarrow$ SelectVariable(X); $X' \leftarrow X - \{x\}$;
3. **while** $D(x) \neq \emptyset$ **do**
4. $d \leftarrow$ SelectValue($D(x)$); $D(x) \leftarrow D(x) - \{d\}$;
5. $S' \leftarrow S \cup \{x \leftarrow d\}$;
6. $D'(x) \leftarrow D(x)$; /* Saving the domain values */
7. **if** ForwardChecking-DTP(X', S') **then**
8. Solve-DTP(X', S');
9. $D(x) \leftarrow D'(x)$; /* Restoring the domain values */
10. **return** *fail*; /* backtracking */

ForwardChecking-DTP(X, S)

1. **forall** $x \in X$ **do**
2. **forall** $d \in D(x)$ **do**
3. **if** *not* Consistency-STP($S \cup \{x \leftarrow d\}$) **then**
4. $D(x) \leftarrow D(x) - \{d\}$;
5. **if** $D(x) = \emptyset$ **then return** *false*; /* dead-end */
6. **return** *true*.

Figure 2: Basic algorithm for solving a DTP. The input is the set X of the meta-variables in the meta CSP of the DTP, and a (partial) solution S of the meta CSP. $D(x)$ is a global variable storing the current domain of the meta variable x.

As observed in [Stergiou & Koubarakis, 2000; Tsamardinos & Pollack, 2003], a DTP can be seen as a "meta CSP", where the variables are the constraints, and the values of the meta-variables are the disjuncts forming the constraints. The constraints of the meta CSP are not explicitly stated. Instead, they are implicitly defined as follows: an assignment θ of values to the meta-variables satisfies the constraints of the meta CSP iff θ forms a consistent STP (an induced STP of the DTP). A solution of the meta CSP is a complete induced STP of the DTP.

Figure 2 shows an algorithm for solving the meta CSP of a DTP [Tsamardinos & Pollack, 2003], which is a variant of the forward-checking backtracking algorithm for solving general CSPs. By appropriately choosing the next meta-variable to handle (function SelectVariable) and its value (function SelectValue), we can show that the algorithm finds a solution (if one exists) with *no backtracking*. Moreover, by a simple modification of the basic algorithm, we can derive an algorithm that is backtrack free even when the meta CSP has no solution. This can be achieved by exploiting the information in the LA-graph $\mathcal{A}$ of the TDA-graph for decomposing its DTP $\mathcal{D}$ into a sequence of "growing DTPs". I.e.,

$$\mathcal{D} = \mathcal{D}_{last} \supset \mathcal{D}_{last-1} \supset ... \supset \mathcal{D}_1,$$

where (i) *last* is the number of the levels in $\mathcal{A}$, (ii) the variables V_i of $\mathcal{D}_i$ ($i = 1..last$) are all the variables of $\mathcal{D}$ corresponding to the action nodes in $\mathcal{A}$ up to level i, and (iii) the constraints of $\mathcal{D}_i$ are all the constraints of $\mathcal{D}$ involving only

variables in V_i. From the decomposed DTP, we can derive an ordered partition of the set of meta-variables X in the meta CSP of the original DTP

$$X = X_1 \cup X_2 \cup ... \cup X_{last},$$

where X_i is the set of the meta-variables corresponding to the constraints in $\mathcal{D}_i - \mathcal{D}_{i-1}$, if $i > 1$, and in $\mathcal{D}_1$ otherwise.

This ordered partition is used to define the order in which SelectVariable chooses the next variable to handle, which is crucial to avoid backtrack: every variable with a single domain value (i.e., an ordering constraint or duration constraint) is selected before every variable with more than one possible value (i.e., a scheduling constraint with more than one time window); if $x_i \in X_i$, $x_j \in X_j$ and $i < j$, then x_i is selected before x_j.

Also the order in which SelectValue chooses the value for a meta-variable is important: given a meta-variable with more than one value, we choose the value corresponding to the earliest available time window. E.g., if the current domain of the meta-variable is

$$\bigcup_{i=1..m} \{(a^+_{start} - a^- \leq -k^-_i) \wedge (a^+ - a^+_{start} \leq k^+_i)\},$$

then SelectValue chooses the j-th value (time window) such that $|k^-_j| < |k^-_h|$, for every $h \in \{1, ..., m\}$, $h \neq j$.

By using these techniques for selecting the next variable to handle and its domain value in the algorithm of Figure 2, we can derive the following result.[5]

Theorem 1 *Given a DTP $\mathcal{D}$ for a TDA-graph, if the meta CSP $\mathcal{X}$ of $\mathcal{D}$ is solvable, then* Solve-DTP *finds a solution of $\mathcal{X}$ with no backtracking. Moreover, this solution is an optimal induced STP of $\mathcal{D}$ for a^-_{end}.*

As a consequence of the previous theorem, we have that, if Solve-DTP performs backtracking (step 10), then the DTP under consideration has no solution. Thus, we can obtain a backtrack free algorithm by replacing step 10 with

10. **stop** and **return** *fail*.

It is easy to see that in the modified algorithm, called Solve-DTP$^+$, every variable is instantiated at most once with the same value. It follows that, under the assumptions that we have a constant maximum number of action preconditions and, for every scheduling constraint, a constant maximum number of windows, the total runtime complexity of Solve-DTP$^+$ is polynomial.

Theorem 2 *Given a TDA-graph $\mathcal{G}$ with DTP $\mathcal{D}$,* Solve-DTP$^+$ *processes the meta CSP corresponding to $\mathcal{D}$ in polynomial time with respect to the number of action nodes in $\mathcal{G}$.*

The actual algorithm that we developed for our planner to find an induced STP for the DTP of a TDA-graph contains some improvements making it more efficient. For lack of space and simplicity of presentation, we omit a detailed description of the improved algorithm, and we indicate only the main differences, which are the following ones:

- the consistency of the STP formed by the values of all the variables of the meta CSP $\mathcal{X}$ with single-valued domains can be checked at the beginning of Solve-DTP, using a single-source shortest-path algorithm: if such an STP is inconsistent, then $\mathcal{X}$ has no solution;
- forward checking is performed only once for each meta-variable: if the first value chosen by SelectValue is not feasible (i.e., ForwardChecking-DTP returns false), then $\mathcal{X}$ has no solution, and thus we can stop the algorithm;
- finally, the improved algorithm is incremental since, as we will see in the next section, at each search step the DTP of the TDA-graph is updated as a consequence of adding a new action node to the graph, or removing an existing one.

Moreover, in order to use the local search techniques described in the next section, we need another change to the basic algorithm: when the algorithm detects that $\mathcal{X}$ has no solution, instead of returning failure, (i) it processes the next meta variables, and (ii) when it terminates, it returns the (partial) induced STP $\mathcal{S}$ formed by the STP-constraints of the DTP and the values assigned to the meta-variables. The optimal solution of $\mathcal{S}$ defines the $\mathcal{T}$-assignment of the TDA-graph.

In the next section, $\mathcal{S}_\mathcal{G}$ denotes the induced STP for the DTP of a TDA-graph $\mathcal{G}$ computed by our method.

3 Local Search Techniques for TDA-Graphs

A TDA-graph $\langle \mathcal{A}, \mathcal{T}, \mathcal{P}, \mathcal{C} \rangle$ may contain two types of *flaw*: unsupported precondition nodes in $\mathcal{A}$ (*propositional flaw*s), action nodes in $\mathcal{A}$ that are unscheduled in $\mathcal{T}$ (*temporal flaw*s). If a level of $\mathcal{A}$ contains a flaw, we say that this level is flawed. A TDA-graph with no flawed level represents a valid plan, and it is called a *solution graph*.

In this section, we present new heuristics for searching a solution graph in the space of TDA-graphs. These heuristics are used to guide a local search procedure, called Walkplan, that was originally proposed in [Gerevini, et al., 2003], and that is the heart of search engine of our planner.

The initial TDA-graph contains only a_{start} and a_{end}. Each search step identifies the neighborhood $N(\mathcal{G})$ (successor states) of the current TDA-graph $\mathcal{G}$ (search state), which is a set of TDA-graphs obtained from $\mathcal{G}$ by adding a *helpful action node* or removing a *harmful action node* in the attempt to repair the *earliest* flawed level of $\mathcal{G}$.[6] In the following, a_i denotes an action node a at level i of $\mathcal{A}$, and l_a the level of a.

Given a flawed level l of $\mathcal{G}$, an action node a_i is *helpful* for l if its insertion into $\mathcal{G}$ at a level $i \leq l$ removes a propositional flaw at l; a_i is *harmful* for l if its *removal* from a level $i \leq l$ of $\mathcal{G}$ (i) would remove a propositional flaw at l, or (ii) would decrease the $\mathcal{T}$-value of a_l, if a_l is unscheduled (intuitively, a_l is unscheduled if $\mathcal{C}$ forces it to start "too late").

The addition/removal of an action node a requires us to update the DTP of $\mathcal{G}$ by adding/removing some ordering constraints between a and other actions in the LA-graph of $\mathcal{G}$, the duration constraints of a, and the scheduling constraint of a (if any). From the updated DTP $\mathcal{D}$, we can use the method described in the previous section to revise $\mathcal{T}$, and to compute a possibly new schedule of the actions in $\mathcal{G}$ (i.e., an optimal solution of $\mathcal{S}_\mathcal{G}$).

The elements in $N(\mathcal{G})$ are evaluated using a *heuristic evaluation function* E consisting of two weighted terms, estimating the additional *search cost* and *temporal cost* of the el-

[5] For lack of space, the proofs are omitted; they are available in an extended version of this paper [Gerevini, et al., 2005].

[6] When we add an action node, the graph is extended by one level, and when we remove an action node, it is "shrunk" by one level. More details in [Gerevini, et al., 2003].

ements (i.e., the number of search steps required to find a solution graph and the plan makespan, respectively). An element with the lowest cost is then selected from $N(\mathcal{G})$ using a "noise parameter" randomizing the search to escape from local minima [Gerevini, et al., 2003]. For lack of space, in the rest of this section we focus only on the search cost term of E.

The search cost of adding an helpful action node a to $\mathcal{G}$ is estimated by constructing a *temporal relaxed plan* π achieving (1) the unsupported precondition nodes of a, (2) the propositional flaws remaining at l after adding a, and (3) the supported precondition nodes of other action nodes in $\mathcal{G}$ that would become *un*supported by adding a. Moreover, we count the number of: (4) action nodes that would become *un*scheduled by adding a to $\mathcal{G}$, (5) unsatisfied timed preconditions of a, (6) actions of π with a scheduling constraint that we estimate cannot be satisfied in the context of $\mathcal{G}$. The search cost of adding a to $\mathcal{G}$ is the number of actions in π plus (4), (5) and (6).

The evaluation of a TDA-graph derived by *removing* an harmful action node a is similar, with π achieving the precondition nodes supported by a that would become *un*supported by removing a and, when l_a precedes the flawed level l under reparation, the unsupported precondition nodes at level l that would not become supported by removing a.

π is constructed using a polymomial backward process similar to the algorithm proposed in [Gerevini, et al., 2003], giving in output two values: a set of actions forming a (sub)relaxed plan, and its estimated earliest finishing time. The initial state I is the state obtained by applying the actions of $\mathcal{G}$ up to level $l_a - 1$, ordered according to their levels.

The main difference in the extended algorithm concerns the choice of the actions forming the relaxed plan. The action b chosen to achieve a (sub)goal g is an action minimizing the sum of (i) the estimated minimum number of additional actions required to support its propositional preconditions, (ii) the number of supported precondition nodes in the LA-graph that would become unsupported by adding b to $\mathcal{G}$, (iii) the number of timed preconditions of b that we estimate would be unsatisfied in $\mathcal{G}$ extended with π ($TimedPre(b)$); and (iv) the number of action nodes scheduled in $\mathcal{T}$ that we estimated would become unscheduled by adding b to $\mathcal{G}$ ($TimeThreats(b)$). (i)-(ii) are computed as described in [Gerevini, et al., 2003]; (iii)-(iv) are new components of the action selection method, and they are computed as follows.

In order to compute $TimedPre(b)$, we estimate the earliest start time of b ($Est(b)$) and the earliest finishing time of b ($Eft(b)$). Using these values, we count the number of the timed preconditions of b that cannot be satisfied. $Eft(b)$ is $Est(b) + Dur(b)$, while $Est(b)$ is the maximum over

- the lowest earliest start time of b computed by an extension of the reachability analysis algorithm given in [Gerevini, et al., 2003], which derives a lower bound on the start time of each domain action;
- the $\mathcal{T}$-values of the action nodes c_i, with $i < l_a$, that are mutex with b (because the addition of b to $\mathcal{G}$ would determine the addition of $c_i^+ - b^- \leq 0$ to $\mathcal{G}$);
- the maximum over an estimated lower bound on the time when all the preconditions of b are achieved in relaxed plan (this estimate is computed from the causal structure of the relaxed plan, the duration and scheduling constraints of its actions, and the $\mathcal{T}$-values of the facts in the initial state I).

In order to compute *TimeThreats*(b), we use a notion of *time slack* between action nodes.

Definition 3 *Given two action nodes $a1$ and $a2$ of a TDA-graph $\langle \mathcal{A}, \mathcal{T}, \mathcal{P}, \mathcal{C} \rangle$ such that $\mathcal{C} \models a1^+ < a2^-$, $slack(a1, a2)$ is the maximum time by which the $\mathcal{T}$-value of $a1^-$ can be consistently increased in $\mathcal{S}_\mathcal{G}$ without violating the time window chosen for scheduling $a2$.*

To estimate whether b is a time threat for an action node a_k ($l \leq k$), we check if $\Delta(\pi_b, a_l) > Slack(a_l, a_k)$ holds, where π_b is the portion of the relaxed plan computed so far, and $\Delta(\pi_b, a_l)$ estimates the delay of the start time of a_l that the addition of the actions in π_b to $\mathcal{G}$ would determine.

4 Experimental Results

We have implemented our approach in a planner called LPG-td, which obtained the 2nd prize in the suboptimal metric-temporal track of the 4th International Planning Competition (IPC-4). LPG-td performed especially well in the domain variants with timed initial literals, in terms of both CPU-time to find a plan and quality of the best plan computed with a CPU-time limit of 30 minutes (LPG-td is an incremental planner finding a succession of valid plans). In this section, we present some experimental results using the test problems of IPC-4.[7] The problems in the `Airport` domain specify at most 6 time windows for each timed precondition, the problems in the `Satellite` domain at most 3 windows, while those in the other domains only one time window. Additional results are available from the web sites of our planner and of IPC-4, and in a technical report including an experimental analysis on solving problems with many windows associated with the timed preconditions [Gerevini, et al., 2005].

Figure 3 shows the CPU-time of LPG-td in three IPC-4 domains with respect to the best among the other three planners of IPC-4 that support timed initial literals: SGPLAN, P-MEP, and TILSAPA.[8] In these domains, LPG-td is generally faster than the other planners and solves more problems.

Table 1 gives a summary of the results for all the IPC-4 domain variants with timed initial literals (252 test problems in total). We compare LPG-td's results with the best results over the corresponding results of *all* the other IPC-4 planners ("AllOthers"). In general, LPG-td solves more problems than AllOthers; the percentage of problems in which it is faster is higher than the one in which it is slower; and the percentage in which it produces better quality plans is much higher.

Finally, it is worth noting that, if in the CPU-time comparison we consider only problems where LPG-td is at least one order of magnitude faster (slower) than AllOthers, then the results in the 3rd column of Table 1 are even more favorable to LPG-td. LPG-td is faster in 31% of the problems, and it is slower in 13% of the problems.

[7] All tests were conducted on an Intel Xeon(tm) 3 GHz, 1 Gbytes of RAM. For a description and formalization of the IPC-4 benchmark problems and domains, see `http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/index.html`.

[8] An abstract of every IPC-4 planner is available in [Edelkamp, et al., 2004]). LPG-td and TILSAPA are the only planners of IPC-4 that addressed the variant of `PipesWorld` with timed initial literals; TILSAPA did not address `UMTS-flaw` with timed initial literals.

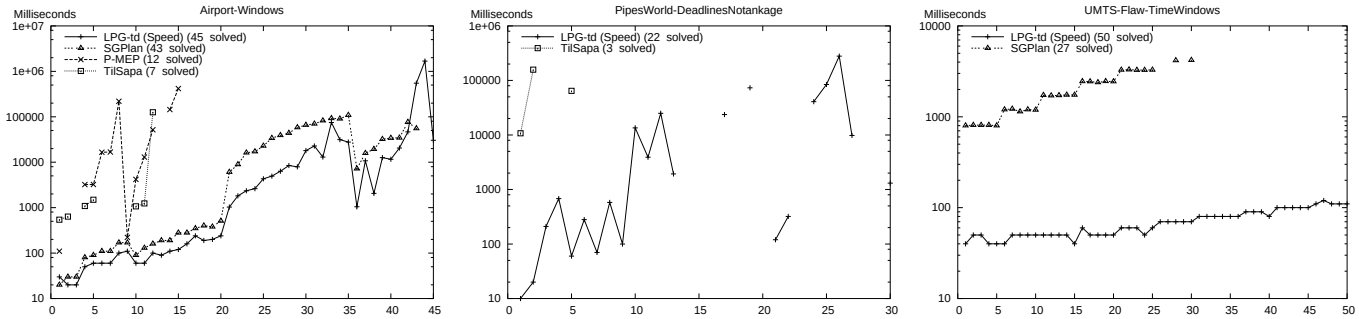

Figure 3: CPU-times of LPG-td, P-MEP, SGPLAN, and TILSAPA in three IPC-4 test domains with timed initial literals. On the x-axis we have the problem names simplified by numbers. On the y-axis, we have the CPU-times in logarithmic scale.

IPC-4 domain	Problems solved	CPU-time better (worse)	Plan quality better (worse)
Airport	90 (86)	86 (4)	58 (5)
PipesWorld	73 (10)	73 (0)	67 (0)
Sat-Complex	53 (67)	6 (67)	71 (12)
Sat-Time	53 (67)	6 (64)	59 (35)
UMTS-Flaw	100 (54)	100 (0)	82 (0)
UMTS	100 (100)	0 (88)	100 (0)
Total	81.3 (67.8)	47.2 (39.3)	73.2 (6.4)

Table 1: A comparison of LPG-td and the best over the results of *all* the other IPC-4 planners. Summary results in terms of: % of problems solved by LPG and AllOthers (in brackets); % of problems in which LPG-td is faster (slower in brackets); % of problems in which LPG-td produces a plan with shorter makespan (longer in brackets).

5 Conclusions

We have presented an approach to temporal planning for domains where actions have durations and must be executed during certain time windows. This allows us to deal with deterministic exogenous events, which is important in many real-world planning domains. Our approach combines constraint-based temporal reasoning and a recent graph-based method for planning. We propose a new plan representation and search space, a polynomial algorithm for temporal constraint reasoning during search, and some local search techniques for planning that exploit temporal information. An analysis of the IPC-4 results show that our planner performs very well compared to other recent temporal planners. We believe that our temporal reasoning results can be exploited also in the context of other approaches to planning.

Like our planner, SAPA uses a relaxed plan heuristic to guide the search [Do et al., 2004]. However, SAPA uses a time slack analysis for selecting these actions that is limited to the actions of the relaxed plan, while our heuristics consider also the actions of the "real" plan under construction. Other very recent planners supporting time windows include DT-POP and MIPS. Edelkamp proposes a method for handling timed preconditions in MIPS with only one time window [Edelkamp, 2004]. DT-POP extends POP-planning with DTPs [Schwartz & Pollack, 2004]. DT-POP supports more temporal features than LPG-td, but it is less efficient, and it does not exploit the plan representation that we use for achieving tractable temporal reasoning during planning. For a more detailed discussion of related work see [Gerevini, et al., 2005].

References

[Blum & Furst, 1997] Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90.

[Do et al., 2004] Do, M., B., Kambhampati, S., and Zimmerman, T. 2004. Planning - Scheduling Connections through Exogenous Events. In *Proc. of WIPIS-04*.

[Dechter, et al., 1991] Dechter, R., Meiri, I., and Pearl, J., 1991. Temporal Constraint Networks, *Artificial Intelligence* 49:61–95.

[Edelkamp, 2004] Edelkamp, S. 2004. Extended Critical Paths in Temporal Planning. In *Proceedings of the ICAPS-04 Workshop on Integrating Planning into Scheduling*.

[Edelkamp & Hoffmann, 2004] Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The Language for the Classic Part of the 4th International Planning Competition. Technical Report 195, Institut für Informatik, Freiburg, Germany.

[Edelkamp, et al., 2004] In Edelkamp, S., Hoffmann, J., Littman, M., Younes, H. (Eds.) 2004. *In Abstract Booklet of the Competing Planners of ICAPS-04*.

[Fox & Long, 2003] Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR* 20:61–124.

[Fox & Long, 2004] Fox, M. and Long, D. and Halsey, K., 2004. An Investigation into the Expressive Power of PDDL2.1. In *Proc. of ECAI-04*.

[Gerevini & Cristani, 1997] Gerevini, A., and Cristani, M., 1997. On Finding Solutions in Temporal Constraint Networks. In *Proc. of IJCAI-97*.

[Gerevini, et al., 2003] Gerevini, A., Saetti, A., and Serina, I. 2003. Planning through Stochastic Local Search and Temporal Action Graphs. *JAIR* 20:239–290.

[Gerevini, et al., 2005] Gerevini, A., Saetti, A., and Serina, I. 2005. An Approach to Temporal Planning in Domains with Deterministic Exogenous Events. Technical Report RT 2005-06-45, DEA, Universitá di Brescia, Italy.

[Laborie & Ghallab, 1995] Laborie, P. and Ghallab, M. 1995. Planning with Sharable Resource Constraints. *Proc. of IJCAI-95*

[Muscettola, 1994] Muscettola, N. 1994 HSTS: Integrating Planning and Scheduling. *Intelligent Scheduling*. Morgan Kaufmann.

[Schwartz & Pollack, 2004] Schwartz, P., J. and Pollack, M., E. 2004. Planning with Disjunctive Temporal Constraints. In *Proc. of WIPIS-04*.

[Stergiou & Koubarakis, 2000] Stergiou, K., Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120:81–117.

[Tsamardinos & Pollack, 2003] Tsamardinos, I., Pollack, M., E. 2003. Efficient solution tecniques for Disjunctive Temporal Reasoning Problems. *Artificial Intelligence* 151:43–90.

[Vere, 1983] Vere, S. A. 1983 Planning in time: Windows and durations for activities and goals. *IEEE Transactions on PAMI* 5(3):246–267.