Available online at www.sciencedirect.com
ScienceDirect
Artificial Intelligence 172 (2008) 899–944
Artificial Intelligence
www.elsevier.com/locate/artint

An approach to efficient planning with numerical fluents and multi-criteria plan quality ☆

Alfonso E. Gerevini *, Alessandro Saetti, Ivan Serina

Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia, Via Branze 38, I-25123 Brescia, Italy

Received 31 August 2006; received in revised form 31 December 2007; accepted 4 January 2008

Available online 24 January 2008

Abstract

Dealing with numerical information is practically important in many real-world planning domains where the executability of an action can depend on certain numerical conditions, and the action effects can consume or renew some critical continuous resources, which in PDDL can be represented by numerical fluents. When a planning problem involves numerical fluents, the quality of the solutions can be expressed by an objective function that can take different plan quality criteria into account.

We propose an incremental approach to automated planning with numerical fluents and multi-criteria objective functions for PDDL numerical planning problems. The techniques in this paper significantly extend the framework of planning with action graphs and local search implemented in the LPG planner. We define the *numerical action graph* (NA-graph) representation for numerical plans and we propose some new local search techniques using this representation, including a heuristic search neighborhood for NA-graphs, a heuristic evaluation function based on relaxed numerical plans, and an incremental method for plan quality optimization based on particular search restarts. Moreover, we analyze our approach through an extensive experimental study aimed at evaluating the importance of some specific techniques for the performance of the approach, and at analyzing its effectiveness in terms of fast computation of a valid plan and quality of the best plan that can be generated within a given CPU-time limit. Overall, the results show that our planner performs quite well compared to other state-of-the-art planners handling numerical fluents.

Keywords: Automated planning; Domain-independent planning; Efficient planning; Planning with numerical information; Numerical fluents; Heuristic search for planning

1. Introduction

Handling quantitative information in automated planning is important for addressing real-world planning problems, where the executability of an action can depend on certain numerical conditions, and the action effects can consume or renew some critical resources. For instance, in a typical logistics domain, the vehicles used to transport goods consume quantities of fuel, depending on distances they travel and their fuel consumption rate; some refuel actions

☆ This work is a (very much) revised and significantly extended version of [A. Gerevini, A. Saetti, I. Serina, Planning with numerical expressions in LPG, in: Proceedings 16th European Conference on Artificial Intelligence (ECAI-04), Valencia, Spain, 2004, pp. 667–671].

* Corresponding author.
E-mail addresses: gerevini@ing.unibs.it (A.E. Gerevini), saetti@ing.unibs.it (A. Saetti), serina@ing.unibs.it (I. Serina).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2008.01.002

can be used to increase the available fuel for the vehicles; and refuel actions consume some amounts of the available money.

The first version of the Planning Domain Definition Language (PDDL), the standard language of the international planning competitions [20], supports *numerical fluents* that can be used to represent continuous resources. In a PDDL numerical planning problem, action conditions and goals can involve expressions with numerical fluents, and action effects can modify the values of the problem numerical fluents. The next version of the language, PDDL2.1 [11], emphasizes these numerical features as one of the main aspects for the 2002 international planning competition [36], and it introduces the specification of objective functions for expressing multi-criteria plan quality metrics. For instance, in a logistics domain, some criteria that can affect the quality of a plan are its makespan, the fuel consumed by the vehicles, the quantity of goods successfully transported, the total money spent, the usage of the vehicles, etc. The objective function is a linear combination of different plan quality criteria. The relative impact on the plan quality of these criteria can be modeled by appropriately weighting the objective function terms. The numerical features of PDDL are also important in the successive versions of the language [13,28] and in the benchmarks developed for the corresponding planning competitions [7,29].

Various planning methods and systems supporting numerical fluents have been proposed (e.g., [1,8,9,12,24,27,33, 34,37,38,41,43,48,49]). However, many of them are domain-dependent, are not fast enough to deal with large-scale problems, or do not compute good quality plans with respect to a multi-criteria objective function.

In this paper, we propose a domain-independent approach to *satisficing planning* for PDDL problems with numerical fluents and multi-criteria objective functions. The word "satisficing" was coined by Simon [45] to mean "rational enough", and subsequently it was adopted by the optimization community to mean "good enough". This term has also been adopted by the planning community to indicate planners aimed at computing plans of good quality, but with no guarantee of their optimality [28]. In the recent years, some powerful domain-independent satisficing planners have been proposed (e.g., [5,26,27]). These systems perform very well in terms of CPU-time to compute a plan, but they are not so efficient in terms of plan quality.

Our planning methods significantly extend previously proposed techniques based on action graphs and local search, that originally were developed for simple propositional planning [19]. The action graph data structure was initially defined as a particular class of subgraphs of the planning graph representation [3], and subsequently it has been modified and extended for representing temporal information [15,17]. In this work, we extend the action graph for representing plans with numerical fluents. The new plan representation, called *numerical action graph* (*NA-graph*), is the base data structure exploited by our approach to numerical planning.

Local search is a powerful method for satisficing planning, as demonstrated by some successful planners adopting this search framework that have been awarded at the last four international planning competitions [4,15,26,31,32]. We propose some new local search techniques for planning in a search space of NA-graphs. Starting from an initial NA-graph, the search process transforms it into an NA-graph representing a valid plan through the iterative application of some search steps modifying the graph. The search is guided by new heuristic techniques that exploit the numerical information stored in the current NA-graph for evaluating the possible successor NA-graphs forming the search neighborhood. In order to make the search process more effective, the search neighborhood is heuristically restricted by considering only the most promising elements. The neighborhood evaluation takes account of both the search cost required to reach a valid plan and the quality of the plan that can be reached. Moreover, we propose an incremental method for deriving plans with increasing quality according to the multi-criteria objective function in the (PDDL) planning problem specification. This method is based on search restarts with particular heuristics for re-initializing the search.

Our techniques are implemented in a planner that took part in the 2004 international planning competition [28] showing good performance especially in the metric and metric-temporal domains involving numerical fluents. In this paper, we experimentally evaluate our approach on a large set of test problems involving numerical fluents and multi-criteria objective functions. We analyze the importance of the proposed techniques for the good performance of the approach, and we compare our system with other recent state-of-the-art planners capable of solving numerical planning problems. The results of this experimental analysis demonstrate the effectiveness of our planning method in terms of CPU-time required to compute a valid plan and quality of the best plan that is generated within a given CPU-time limit.

The paper is organized as follows. Section 2 presents the underlying formal model of planning problems with numerical fluents and multi-criteria objective functions. Section 3 introduces the numerical action graph representa-

tion that we use for solving these problems. Section 4 describes the basic local search procedure, the definition and evaluation of the search neighborhood, and the techniques for the incremental computation of good quality plans. Section 5 concerns dealing with PDDL durative actions and plan makespan optimization in numerical planning problems. Section 6 presents our experimental analysis. Section 7 discusses related work, including our previous work. Finally, Section 8 gives the conclusions and indicates future work.

2. Preliminaries: Planning with numerical information

Planning with numerical fluents and multi-criteria objective functions addresses problems with continuous resources and multiple criteria to evaluate the quality of the plans that are generated. Many real-world problems involve continuous resources, such as fuel, money, time, space, etc. In a simple propositional (STRIPS or ADL-like) setting, these resources can be approximated by treating them as discrete resources (with a finite set of reachable values), which can be modeled by a set of different predicates or constants. The drawback of this approach is that the set of resource values that are reachable in a fixed number of search steps can grow exponentially in this number [30].

In this section, first we formalize the semantic model of planning with numerical fluents used in our approach, which is based on the semantics of PDDL2.1 [11] (with some minor differences). Then, we briefly describe its connection to the syntax of PDDL2.1. Differently from Fox and Long's semantics for planning with numerical fluents in PDDL2.1, our characterization of numerical planning is independent from the syntax of the planning language, and, in this sense, we believe it is simpler. Moreover, our model adopts less restrictive conditions for action concurrency. Finally, in our formalization, a set of actions that are *not* executable in a world state s (because one or more of their preconditions are not satisfied) produces a successor state of s that, according to Fox and Long's formalization, would be undefined. This will be used in the plan representation and heuristics exploited by our planning approach, which searches in a space of partial plans possibly containing non-executable actions.

Also Helmert proposed a general and compact formalization of numerical planning [25], which has several differences with ours and serves as a base for different purposes. His model cannot be directly adopted as a formal base for the techniques presented in this paper, since it does not explicitly model plans with concurrent actions, plans with non-executable actions, and planning problems with multi-criteria objective functions. In addition, we use some different notation, terminology and assumptions that we believe are more suitable for our purposes.

A *numerical fluent* is a state variable over the set $\mathbb{R}$ of real numbers such that there exists at least one domain action that can change its initial value specified in the problem initial state. Our formalization of planning problem with numerical fluents, that we call *numerical planning problem*, is derived by extending the state-transition model of the "classical planning problem" (for a description of this model see, e.g., [21]).

In the rest of the paper, L denotes the set $\{\ell_1, \ldots, \ell_m\}$ of positive literals representing the propositional fluents of the planning problem, while X denotes the set $\{x_1, \ldots, x_n\}$ of numerical variables representing the numerical fluents of the planning problem. Without loss of generality, we assume that the elements of X are ordered according to an arbitrary order. $\overline{X}$ denotes the n-tuple of all the ordered numerical variables.

Definition 1 *(World state).* Let L be a set of positive literals and $X = \{x_1, \ldots, x_n\}$ a set of numerical variables (fluents). A world state for L and X is a pair $\langle I, \bar{x}\rangle$, where $I \subseteq L$ and $\bar{x} = \langle q_1, \ldots, q_n\rangle$ is a tuple of real numbers (i.e., $\bar{x} \in \mathbb{R}^n$) representing the assignment $x_i = q_i$, for $i = 1 \ldots n$.

A world state for a set of propositional fluents and a set of numerical fluents specifies the propositional fluents that are true (i.e., those in I), and a value for each numerical fluent (specified by the tuple $\bar{x}$). Under the closed world assumption, if a propositional fluent is not in I, then in that state it is false.

In numerical planning problems, actions can have *numerical preconditions* that affect their executability, and *numerical effects* that change the values of some numerical fluents in the world states resulting by applying the actions. Since the negative propositional preconditions of an action and the negative problem goals can be compiled into positive preconditions and goals obtaining an equivalent numerical planning problem (see, e.g., [18,35]), without loss of generality, we assume that the planning problem has only positive propositional preconditions and goals. We use "$\bot$" and "$\top$" to denote *false* and *true*, respectively.

Definition 2 *(Action with numerical information).* Let L be a set of propositional positive literals and X a set of n variables over the real numbers. An action with numerical information is a tuple

$$a = \langle PP(a), NP(a), PE(a)^+, PE(a)^-, NE(a) \rangle,$$

where

- $PP(a) \subseteq L$ is a set of propositions called propositional (positive) preconditions of a;
- $NP(a) = \{\psi_j^a\}_{j=1..p_a}$ is a set of p_a functions $\psi_j^a : \mathbb{R}^n \to \{\bot, \top\}$ *called* numerical preconditions of a such that

$$\psi_j^a(\bar{x}) = \begin{cases} \top & \text{if } f_j^a(\bar{x})\{<, \leqslant, \geqslant, >\}0 \\ \bot & \text{otherwise,} \end{cases}$$

  where f_j^a is a rational function, and the function domain tuple of ψ_j^a is an assignment of real numbers to the n variables in X;
- $PE(a)^+ \subseteq L$ is a set of propositions called *propositional positive effects of* a;
- $PE(a)^- \subseteq L$ is a set of propositions called *propositional negative effects of* a;
- $NE(a) = \{\xi_j^a\}_{j=1..q_a}$ is a set of q_a rational functions $\xi_j^a : \mathbb{R}^n \to \mathbb{R}^n$ called *numerical effects of* a, where the function domain and codomain tuples are assignments of real values to the n variables in X, and each $\xi_j^a(\bar{x})$ differs from $\bar{x}$ by at most one value.

We will assume that the functions defining the numerical action preconditions and effects are *monotonic* in each variable separately. The motivation of this assumption, which, although not made in PDDL2.1, is also adopted by other well-known prominent systems for numerical planning (e.g., [4,9,27]) and is consistent with all existing benchmarks of the planning competitions, will be discussed in the section dedicated to our search techniques.

As a running example, we will use a simple problem in the known ZenoTravel domain [36,39], concerning moving people in a network of locations by using aircrafts consuming fuel. We will consider $L = \{\texttt{at(A, L1)}, \texttt{at(A, L2)}, \texttt{at(P, L1)}, \texttt{at(P, L2)}, \texttt{in(P, A)}\}$ and $\overline{X} = \langle \texttt{fuel(A)}, \texttt{fuel\_used} \rangle$, where A is an aircraft, L1 and L2 are locations, and P is a person.

Example 1 *(Action with numerical information).* A possible action for the problem of our running example is $a = \texttt{fly(A, L1, L2)}$, which is defined as follows

- $PP(\texttt{fly(A, L1, L2)}) = \{\texttt{at(A, L1)}\}$;
- $NP(\texttt{fly(A, L1, L2)}) = \{\psi_1^a(\texttt{fuel(A)}, \texttt{fuel\_used})\}$, where $\psi_1^a(\texttt{fuel(A)}, \texttt{fuel\_used})$ is equal to $\top$, if $\texttt{fuel(A)} \geqslant 100$, and equal to $\bot$ otherwise;
- $PE(\texttt{fly(A, L1, L2)})^+ = \{\texttt{at(A, L2)}\}$; $PE(\texttt{fly(A, L1, L2)})^- = \{\texttt{at(A, L1)}\}$;
- $NE(\texttt{fly(A, L1, L2)}) = \{\xi_1^a(\texttt{fuel(A), fuel\_used}), \xi_2^a(\texttt{fuel(A), fuel\_used})\}$
  $= \{\langle \texttt{fuel(A)} - 100, \texttt{fuel\_used} \rangle, \langle \texttt{fuel(A)}, \texttt{fuel\_used} + 100 \rangle\}$.

Definition 3 *(Numerical planning problem).* A numerical planning problem is a tuple $\Pi = \langle L, X, A, s_0, PG, NG \rangle$, where

- L is a set of propositional positive literals representing the propositional fluents of Π;
- X is a set of n real variables representing the set of numerical fluents of Π;
- A is a set of actions with numerical information involving the variables in X;
- $s_0 \in 2^L \times \mathbb{R}^n$ is the *initial world state* of Π;
- $PG \subseteq L$ is a set of propositions called the *propositional* (*positive*) *goals* of Π;
- $NG = \{\psi_j^\Pi\}_{j=1..n_\Pi}$ is a set of n_Π functions, called the *numerical goals* of Π, which are defined analogously to the numerical action preconditions in Definition 2.

Two actions can be concurrently planned if they do not interfere. We adopt the same definition of *interference between actions* given by Fox and Long for PDDL2.1 [11], with the following two exceptions:[1] we do not consider a pair of actions interfering when (i) an action asserts a (positive) precondition of the other one, or (ii) the actions have numerical effects scaling up (down) the value of the same numerical fluent. For instance, concerning point (ii), if action a has a numerical effect scaling up x by 2, and action b has a numerical effect scaling up x by 3, then in a valid PDDL2.1 plan a and b cannot be concurrently executed, while in our context they can be parallelized. It is easy to see that a and b can be serialized in any order producing the same state, and hence we decided to treat these actions as non-interfering.

As in Fox and Long's definition, we consider two actions a and b *non*-interfering *only if*:

(1) the propositional negative effects of a (b) falsify no positive precondition of b (a),
(2) the propositional negative effects of a (b) falsify no propositional positive effects of b (a),
(3) the composition of the numerical effects of a (b) and b (a) is commutative,
(4) there is no interaction between the numerical preconditions of a (b) and the numerical effects of b (a), i.e.,

$$\forall \psi^a \in NP(a) \cdot \forall \xi^b \in NE(b) \cdot \forall \bar{x} \in \mathbb{R}^n, \quad \psi^a\big(\xi^b(\bar{x})\big) = \psi^a(\bar{x}), \quad \text{and}$$
$$\forall \psi^b \in NP(b) \cdot \forall \xi^a \in NE(a) \cdot \forall \bar{x} \in \mathbb{R}^n, \quad \psi^b\big(\xi^a(\bar{x})\big) = \psi^b(\bar{x}).$$

A set of non-interfering actions planned at the same time is called a *happening* [11]. A happening in a world state s causes some changes to s, which can be specified by a *state transition function* γ. Function γ is defined as $\gamma : \Sigma \times 2^A \to \Sigma$, where $\Sigma \subseteq 2^L \times \mathbb{R}$ is the set of possible world states, and A is the set of actions in the numerical planning problem. Function γ defines how a world state $s \in \Sigma$ changes because of a happening $h \in 2^A$. More precisely, the state $\gamma(\langle I, \bar{x}\rangle, h)$ is defined as follows

$$\gamma\big(\langle I, \bar{x}\rangle, h\big) = \left\langle \bigcup_{a \in h} \big(I \cup PE(a)^+ \setminus PE(a)^-\big), \xi_1 \circ \cdots \circ \xi_m(\bar{x}) \right\rangle$$

where $\xi_j \in \bigcup_{a \in h} NE(a)$, for $j = 1 \ldots m$, $m = \sum_{a \in h} |NE(a)|$, "$\circ$" denotes the composition operator, and $|\cdot|$ denotes the set cardinality operator. Note that the numerical effect functions can be composed according to any total order of such functions, because of condition (3) for action non-interference. Moreover, the state obtained by applying a set of actions h to a world state is well defined even if not all actions in h have every precondition satisfied in s.

A *plan* is a sequence of happenings. The *empty plan* is the empty sequence of happenings. The final state obtained by executing a non-empty plan $\pi = \langle h_1, \ldots, h_k\rangle$ in a world state s is the state obtained by applying the happenings $h_1, \ldots, h_k$ of π to s, according to their order in π. The *state-transition function* Γ of a plan π from a world state s is defined as follows

$$\Gamma(s, \pi) = \begin{cases} s & \text{if } \pi \text{ is empty} \\ \Gamma(\gamma(s, h_1), \langle h_2, \ldots, h_k\rangle) & \text{if } \pi = \langle h_1, \ldots, h_k\rangle. \end{cases}$$

Let $\pi = \langle h_1, \ldots, h_k\rangle$ be a plan for a numerical planning problem Π, and s_0 the initial world state of Π. The sequence of world states $\mathcal{S} = \langle s_0, s_1, \ldots, s_k\rangle$ such that $s_i = \langle I_i, \bar{x}_i\rangle = \gamma(h_i, s_{i-1})$, for $i = 1 \ldots k$, is the *state trajectory* of π from s_0. We say that *plan π is executable* in s_0 if and only if, $\forall\, h_i \in \pi$ and $\forall a \in h_i$, *action a is executable* in the state $s_{i-1} \in \mathcal{S}$, i.e., $PP(a) \subseteq I_{i-1}$ and $\forall \psi^a \in NP(a),\ \psi^a(\bar{x}_{i-1}) = \top$. The empty plan is a special case of an executable plan for any planning problem.

Let $\pi = \langle h_1, \ldots, h_k\rangle$ be a plan for a numerical planning problem Π, and s_k the final state of π executed in the problem initial state s_0. We say that *plan π is valid* for Π if and only if π is executable in s_0 and all problem goals are satisfied in s_k. We call a possibly non-valid plan for Π a *partial plan* for Π.

[1] Because of these exceptions, our definition is less restrictive than the definition of Fox and Long. However, this is not a strong requirement for our approach. Adopting exactly the same definition of Fox and Long and revising the techniques presented in this paper accordingly would be straightforward.

Definition 4 *(Numerical planning problem with objective function).* A numerical planning problem with objective function is a pair $\Pi^+ = \langle \Pi, \Phi^\Pi \rangle$ such that Π is a numerical planning problem and either $\Phi^\Pi = \min\{g(\overline{X})\}$ or $\Phi^\Pi = \max\{g(\overline{X})\}$, where g is a real n-ary function and X is the set of real variables in Π.

For every tuple $\bar{x}$, we have that $\max\{g(\bar{x})\} = -\min\{-g(\bar{x})\}$. Hence, without loss of generality, we can treat every objective function maximizing $g(\overline{X})$ as the negation of the objective function minimizing $-g(\overline{X})$. In the rest of the paper, we use this assumption and we call the min-argument f of the objective function the *plan metric function*, i.e., either $f = g(\overline{X})$ or $f = -g(\overline{X})$. Moreover, consistently with the existing benchmarks of the planning competitions and other systems supporting numerical planning with multi-criteria objective function (e.g., [2,8]), we assume that the plan metric function is a linear polynomial.

The plan metric function f of a numerical planning problem Π defines *the quality of a* (*partial*) *plan* π for Π as a function over the values of the numerical fluents in the final world state $s_G = \Gamma(s_0, \pi)$. Let $\langle I_1, \bar{x}_1 \rangle = \Gamma(s_0, \pi_1)$ and $\langle I_2, \bar{x}_2 \rangle = \Gamma(s_0, \pi_2)$ be the final states of two plans π_1 and π_2 for Π, respectively. The quality of plan π_1 is better than the quality of plan π_2 if $f(\bar{x}_1) < f(\bar{x}_2)$.

The last three versions of the planning domain definition language, PDDL2.1/2.2/3.0 [11,13,28], support planning with numerical fluents and multi-criteria objective functions. In PDDL, numerical fluents are defined through particular positive literals called *primitive numerical expressions*; numerical preconditions are defined through relational expressions constructed by using a *relational operator* ($<$, $\leqslant$, $=$, $\geqslant$, or $>$) between primitive or non-primitive numerical expressions, that are obtained by combining primitive numerical expressions through arithmetic operators;[2] numerical effects are defined through some expressions constructed by using an *assignment operator* (`assign`, `increase`, `decrease`, `scale-up`, or `scale-down`).

For example, the PDDL definition of the ZenoTravel action $a =$ (`fly A L1 L2`) includes the following numerical precondition and numerical effect (`(distance L1 L2)` and `(fuel_burn A)` are static numerical fact that can be simplified to constant values):

```
(>= (fuel A) (* (distance L1 L2) (fuel_burn A)))
(increase (fuel_used) (* (distance L1 L2) (fuel_burn A))).
```

According to our formalization, these PDDL statements are expressed

$$\psi_1^a(\overline{X}) = \top, \quad \text{if } \texttt{fuel(A)} \geqslant \texttt{distance(L1, L2)} \cdot \texttt{fuel\_burn(A)}, \quad \bot \text{ otherwise}$$
$$\xi_1^a(\overline{X}) = \langle \texttt{fuel(A)} - \texttt{distance(L1, L2)} \cdot \texttt{fuel\_burn(A)}, \texttt{fuel\_used} \rangle.$$

Note that the output of the function corresponding to a numerical precondition expressed in PDDL can be affected by only a strict subset of the numerical fluents in the planning problem.

Finally, in PDDL we can express objective functions through the "`:metric` field" in the problem definition file, specifying an arithmetic combination of numerical expressions to be minimized or maximized. For example, the problem description can be augmented with the expression `(:metric minimize (+ (total-time) (* 5 (fuel_used))))` corresponding to the multi-criteria objective function $\Phi^\Pi = \min(\texttt{total-time} + 5 \cdot \texttt{fuel\_used})$, where `total-time` is a special numerical fluent that in PDDL represents the plan makespan.

3. Plan representation: Numerical action graph

In this section, we present a plan representation which is the base of our approach: the *numerical action graph* (NA-graph). The NA-graph is an extension of the *linear action graph* (LA-graph) [15] to include numerical information. An LA-graph can be seen as a variant of the well-known planning graph [3] representation for propositional (STRIPS-like) planning problems. An LA-graph for a planning problem Π is a directed acyclic leveled graph alternating a *fact level* and an *action level*. Each fact level contains a set of *fact nodes*, each of which is labeled by the proposition in L (the set of problem literals) that it represents. Each fact node labeled f at a level l is associated with a *no-op* action

[2] In this paper we do not treat PDDL2.1 preconditions of type "=". Although a =-precondition can be expressed by two preconditions of type "$\leqslant$" and "$\geqslant$", respectively, this trivial indirect way to support equality could be inadequate in our approach. We think that these particular preconditions, which rarely occur in the known benchmark problems, could be better handled in our local search by a dedicated technique.

node at level l, representing a dummy action having proposition f as its only (positive) precondition and effect. Each action level contains one action node labeled by the name of the action in A (the set of the problem actions) 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 (*precondition edges*) from the fact nodes at level l representing the positive preconditions of a (*precondition nodes*), and by outgoing edges (*effect edges*) to the fact nodes at level $l + 1$ representing the effects of a (*effect nodes*).

In an LA-graph, two action nodes are *mutually exclusive* (*mutex*) when the corresponding represented actions are "persistently" mutex [10]. Two actions a and b are persistently mutex if they interfere, or there exists a propositional precondition p of a and a propositional precondition q of b such that in no reachable state both p and q are true.[3] A set of persistent mutex relations can be efficiently precomputed by using an algorithm given in [15].

An LA-graph $\mathcal{A}$ also contains a set $\Omega_{\mathcal{A}}$ of *ordering constraints* between actions in the (partial) plan represented by the graph. These constraints are

- constraints imposed during search to deal with mutually exclusive action nodes: *if an action node a at level l of $\mathcal{A}$ is mutex with an action node b at a level after l, then the action corresponding to a is constrained to finish before the start of the action corresponding to b*;
- constraints between actions implied by the causal structure of the plan: *if an action node a is used to support a precondition node of an action node b, then the action corresponding to a is constrained to finish before the start of the action corresponding to b*.

The set of ordering constraints is updated at each search step as described in [15], which guarantees that the plan contains no threat to the causal relations created during planning. We now give a formal definition of NA-graph and then an example.

Definition 5 *(Numerical action graph).* A numerical action graph (NA-graph) for a numerical planning problem $\Pi = \langle L, X, A, s_0, PG, NG\rangle$ is a triple $\langle \mathcal{A}, \mathcal{M}, \Omega\rangle$ where

- $\mathcal{A}$ is a linear action graph extended as follows:
  - each fact level of $\mathcal{A}$ contains two additional types of node called *numerical precondition node* and *numerical fluent node*;
    - each numerical precondition node of an action node representing an action $a \in A$ is labeled by a PDDL numerical expression representing a numerical precondition in $NP(a)$;
    - each numerical fluent node is labeled by a PDDL primitive numerical expression representing a numerical fluent in X;
  - an action node at a level l of $\mathcal{A}$ representing an action $a \in A$ is connected by (i) additional precondition edges from the numerical precondition nodes at level l, representing the numerical preconditions of a, and (ii) additional effect edges to the numerical fluent nodes at level $l + 1$. Each effect edge from a to a numerical fluent node is labeled by a PDDL numerical expression representing a numerical effect in $NE(a)$.
- $\mathcal{M}$ is an assignment of real values to the numerical fluent nodes of $\mathcal{A}$.
- Ω is a set of ordering constraints between the actions represented in $\mathcal{A}$.

The values associated with the nodes at a level l of an NA-graph define a vector $\bar{x}_l$ of values for $\overline{X}$. We assume that the initial level of any NA-graph contains only the special action node a_{start}, and the last level contains only the special action node a_{end}. The propositional effect nodes of a_{start} represent the propositional facts that are true in the initial world state s_0 of the planning problem Π under consideration; the effect edges of a_{start} to numerical fluent nodes determine the initial values of the numerical fluents of Π as specified in s_0; finally, the precondition nodes of a_{end} represent the propositional goals and the numerical goals of Π.

[3] A world state s is reachable from a state s_0 in a planning problem Π if and only if there exists a plan π for Π such that π is executable in s_0 and $s = \Gamma(s_0, \pi)$. Moreover, note that, while the definition of persistent mutex relations extends the notion of action interference previously given, it is easy to see that, by definition of valid plan, no happening in a valid plan can contain a pair of persistently mutex actions.

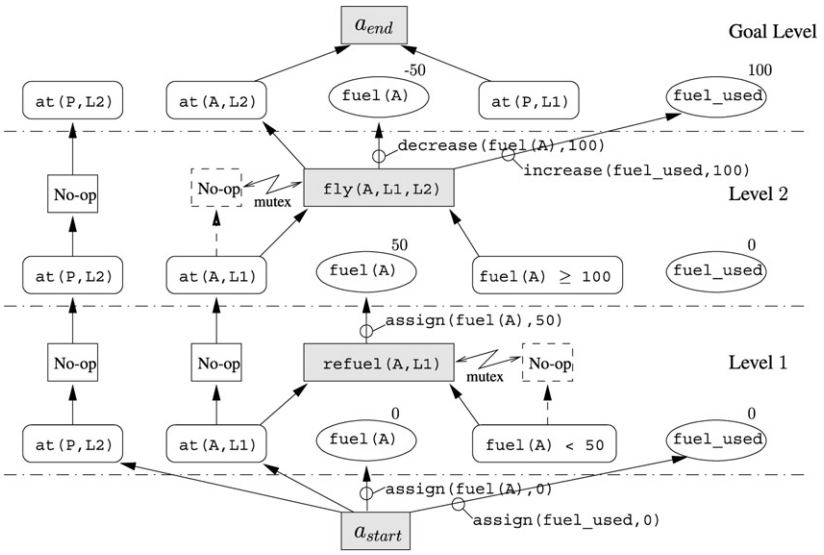


Fig. 1. An NA-graph for a numerical planning problem in the ZenoTravel domain. Grey boxes are action nodes; white boxes are no-op nodes or fact nodes. Elliptical boxes are numerical fluent nodes marked with the corresponding $\mathcal{M}$-values. Action node `fly(A,L1,L2)` blocks the propagation of fact node `at(A,L1)` at level 2, while action node `refuel(A,L1)` blocks the propagation of numerical precondition `fuel(A)<50` at level 1.

In the following, a_l denotes the action represented at level l in the NA-graph under consideration. Each level l of the graph identifies a particular world state that we indicate with $\langle I_l, \bar{x}_l \rangle$. The world state $\langle I_l, \bar{x}_l \rangle$ is obtained by applying the actions in $\mathcal{A}$ up to level $l-1$ to s_0, ordering such actions according to their level in $\mathcal{A}$. More formally,

$$\langle I_l, \bar{x}_l \rangle = \begin{cases} s_0 & \text{if } l = 1, \\ \Gamma(\langle I_1, \bar{x}_1 \rangle, \langle \{a_1\}, \ldots, \{a_{l-1}\} \rangle) & \text{if } l > 1. \end{cases}$$

The value assigned by $\mathcal{M}$ to a numerical fluent node x at level l is equal to the value associated with x in the tuple $\bar{x}_l$.

Example 2 *(Numerical action graph).* Fig. 1 gives an example of NA-graph $\mathcal{A}$ for a numerical planning problem in the ZenoTravel domain. The initial state is $\langle \{\texttt{at(A, L1)}, \texttt{at(P, L2)}\}, \langle 0, 0 \rangle \rangle$; the propositional goal set is $\{\texttt{at(A, L2)}, \texttt{at(P, L1)}\}$ and the numerical goal set is empty. The nodes labeled `refuel(A, L1)` and `fly(A, L1, L2)` are action nodes. The nodes labeled `at(A, L1)`, `at(A, L2)`, `at(P, L1)` and `at(P, L2)` are fact nodes. The nodes labeled `fuel(A)` and `fuel_used` are numerical fluent nodes. The nodes labeled $\texttt{fuel(A)} < 50$ and $\texttt{fuel(A)} \geqslant 100$ are numerical precondition nodes. The world states identified with the three levels of the graph are:

level 0: $\langle \{\texttt{at(A, L1)}, \texttt{at(P, L2)}\}, \langle 0, 0 \rangle \rangle$,
level 1: $\langle \{\texttt{at(A, L1)}, \texttt{at(P, L2)}\}, \langle 50, 0 \rangle \rangle$,
level 2: $\langle \{\texttt{at(A, L2)}, \texttt{at(P, L2)}\}, \langle -50, 100 \rangle \rangle$.

The set of ordering constraints in $\mathcal{A}$ is:

$$\Omega_{\mathcal{A}} = \{a_{start} \prec \texttt{refuel(A, L1)}, a_{start} \prec \texttt{fly(A, L1, L2)}, \texttt{refuel(A, L1)} \prec a_{end}, \texttt{fly(A, L1, L2)} \prec a_{end}, \texttt{refuel(A, L1)} \prec \texttt{fly(A, L1, L2)}\}.$$

For instance, $\texttt{refuel(A, L1)} \prec \texttt{fly(A, L1, L2)} \in \Omega_{\mathcal{A}}$ because the pair of actions involved in the constraint interfere (condition 4 of non-interference is violated) and the graph level of `refuel(A, L1)` precedes the graph level of `fly(A, L1, L2)`.

In an LA-graph the effect nodes can be automatically propagated to the next levels of the graph through the corresponding no-op nodes, until there is an interfering (mutex) action "blocking" the propagation, or the last level of the graph has been reached [15]. Similarly, the supported numerical precondition nodes can be automatically

propagated to the next levels of the graph through the corresponding no-op nodes until there is an action modifying the value of the corresponding numerical precondition from $\top$ to $\bot$ (for an example, see Fig. 1). In the rest of the paper, we assume that every NA-graph incorporates this propagation.

The next two definitions introduce the notion of *linear plan* and *parallel plan* represented by an NA-graph.

Definition 6 *(Linear plan).* Let $\mathcal{A}$ be an NA-graph for a numerical planning problem Π. The linear plan of $\mathcal{A}$ for Π is the plan formed by the actions represented in $\mathcal{A}$ totally ordered according to the levels of the corresponding action nodes in $\mathcal{A}$.

Intuitively, the parallel plan represented by an NA-graph can be constructed from the NA-graph by scheduling the actions represented in the graph to the earliest happening of the plan, according to the action ordering constraints in $\Omega_{\mathcal{A}}$.

Definition 7 *(Parallel plan).* Let $\mathcal{A}$ be an NA-graph for a numerical planning problem Π with action nodes representing a set $S_{\mathcal{A}}$ of actions in Π. The parallel plan of $\mathcal{A}$ for Π is the sequence of happenings $\langle h_1, \ldots, h_n\rangle$ such that

- $h_1 = \{a \in S_{\mathcal{A}} | \neg\exists b \in S_{\mathcal{A}}$ such that $\Omega_{\mathcal{A}} \models b \prec a\}$,
- $h_i = \{a \in S_{\mathcal{A}} | \exists b \in h_{i-1}$ such that $\Omega_{\mathcal{A}} \models b \prec a$ and $\neg\exists c \in (S_{\mathcal{A}} - \bigcup_{j:1..i-1} h_j)$ such that $\Omega_{\mathcal{A}} \models c \prec a\}$, for $i = 2 \ldots n$.

A numerical action graph can contain some *flaws*. A flaw is a precondition node in $\mathcal{A}$ that is not *supported*. A propositional precondition node a at a level l of an NA-graph $\mathcal{A}$ is supported if and only if there is an action node (or a no-op node) at level $l-1$ of $\mathcal{A}$ connected to p by an effect edge. A numerical precondition node at a level l of $\mathcal{A}$ is supported if and only if the corresponding numerical precondition ψ represented by such a node is satisfied in the world state $\langle I_l, \bar{x}_l\rangle$ identified with level l of $\mathcal{A}$, i.e., iff $\psi(\bar{x}_l) = \top$.

A numerical action graph without flaws is called *Solution NA-graph*. The following lemma will be used to prove that every solution NA-graph for a numerical planning problem Π represents a valid (linear or parallel) plan for Π, which is formally stated by the next theorem.

Lemma 1. *Let $\mathcal{A}$ be an NA-graph for a numerical planning problem $\Pi = \langle L, X, A, s_0, PG, NG\rangle$. If the linear plan of $\mathcal{A}$ is a valid plan for Π, then the parallel plan of $\mathcal{A}$ is a valid plan for Π.*

Proof. Suppose that the linear plan π_l of $\mathcal{A}$ is valid for Π, while the parallel plan $\pi_p = \langle h_1, \ldots, h_n\rangle$ of $\mathcal{A}$ is not valid for Π. We show that this leads to a contradiction. Let $\langle s_0, \ldots, s_n\rangle$ be the state trajectory of π_p from s_0. By definition of valid plan, π_p is not executable or at least one problem goal is *false* in s_n. First we consider the case in which π_p is not executable, and then we consider the case in which not all the problem goals hold in s_n. If π_p is not executable, then by definition of executable plan, in π_p there exists an action $a \in h_i$ such that a is not executable in s_{i-1}, $0 < i \leqslant n$. So, by definition of executable action, we have that $PP(a)$ or $NP(a)$ contains a precondition p that is *false* in s_{i-1}.

Suppose that $p \in PP(a)$. We distinguish two cases: p holds in s_0, $\neg p$ holds in s_0. Suppose that p holds in s_0. Since a is not executable in π_p, there exists an action $b \in h_j$ such that $p \in PE(b)^-$, and p stays *false* from s_j to s_{i-1}, for some $j < i$. But then, by construction of $\Omega_{\mathcal{A}}$ (the ordering constraints in $\mathcal{A}$), conditions (1–2) of non-interfering actions and definition of mutex actions, either $a \prec b \in \Omega_{\mathcal{A}}$ or $b \prec a \in \Omega_{\mathcal{A}}$. But $a \prec b \in \Omega_{\mathcal{A}}$ cannot hold because, by Definition 7 and construction of the ordering constraints in an NA-graph, b cannot belong to h_j. Hence, we have that $b \prec a \in \Omega_{\mathcal{A}}$ must hold. Then, since we are assuming that π_l is valid, by construction of $\Omega_{\mathcal{A}}$, in π_l (and π_p) there exists an action c that is used to achieve precondition p of a, and such that $c \prec a \in \Omega_{\mathcal{A}}$ and $p \in PE(c)^+$. Thus, by Definition 7, $c \in h_k$ for some k such that $j < k < i$. This contradicts our assumption that $\neg p$ persists from s_j to s_{i-1}.

Suppose now that $\neg p$ holds in s_0. Then, since π_l is a valid plan for Π, in π_l there exists an action b such that $p \in PE(b)^+$ and $b \prec a \in \Omega_{\mathcal{A}}$ (in π_l b is used to achieve precondition p of a). Moreover, since π_l is valid, for each action $c \in \pi_l$ such that $p \in PE(c)^-$, by construction of $\Omega_{\mathcal{A}}$, either $c \prec b \in \Omega_{\mathcal{A}}$ and $c \prec a \in \Omega_{\mathcal{A}}$ hold, or $b \prec c \in \Omega_{\mathcal{A}}$ and $a \prec c \in \Omega_{\mathcal{A}}$ hold. Since $b \prec a \in \Omega_{\mathcal{A}}$ and we are assuming $a \in h_i \in \pi_p$, by construction of π_p, we have that

$b \in h_j \in \pi_p$ for some $j \in \{1, \ldots, i-1\}$ and that $c \notin h_{j+1} \cup \cdots \cup h_{i-1}$. It follows that p must be *true* in s_{i-1}, and this contradicts our assumption that $p \in PP(a)$ does not hold in s_{i-1}.

Now we consider the case in which $p \in NP(a)$. By conditions (3–4) of non-interfering actions, definition of mutex actions and construction of $\Omega_{\mathcal{A}}$, we have that for any pair of actions a_x and b_x involving a numerical fluent x, a_x and b_x are ordered in $\Omega_{\mathcal{A}}$, or their effects are commutative (the value of x in the state resulting by applying the two actions in parallel or in any sequential order is the same). Thus, the value(s) of the numerical fluent(s) involved in p is (are) the same in s_{i-1} and in the world state where a is executed according to π_l. It follows that there cannot exist a numerical precondition of $a \in h_i \in \pi_p$ that is not satisfied in s_{i-1}.

Thus, we have shown that all actions in π_p are executable, which contradicts our assumption that a is not executable because of p not holding in s_{i-1}.

Concerning the case in which π_p is not valid because of a problem goal that is false in the final state s_n, since the problem goals can be viewed as the preconditions of the special action a_{end}, we can use the same previous argumentation to show that also this case is impossible. □

Theorem 1. *Let $\mathcal{A}$ be a solution NA-graph for a numerical planning problem $\Pi = \langle L, X, A, s_0, PG, NG\rangle$. The linear plan and the parallel plan of $\mathcal{A}$ are valid plans for Π.*

Proof. Let $\pi_l = \langle h_1, \ldots, h_k\rangle$ be the linear plan of $\mathcal{A}$, and $\langle s_0, \ldots, s_k\rangle$ the state trajectory of π_l with $s_k = \langle I_k, \bar{x}_k\rangle$. Suppose that π_l is not a valid plan for Π, then, by definition of valid plan, we have that at least one of the following conditions must hold: (1) π_l is not executable in s_0, (2) $PG \not\subseteq I_k$, (3) $\exists \psi^{\Pi} \in NG$ such that $\psi^{\Pi}(\bar{x}_k) = \bot$. We show that, if any of these conditions held, $\mathcal{A}$ would not be a solution NA-graph, which contradicts the assumption in the theorem.

If (1) held, then by definition of executable plan there would exist a happening h_i in π containing an action a that is not executable in s_{i-1}, for some i such that $0 < i \leqslant k$. Hence, we would have that (a) $PP(a) \not\subseteq I_{i-1}$ or (b) $\exists \psi^a \in NP(a)$ such that $\psi^a(\bar{x}_{i-1}) = \bot$ holds. It follows that, by construction of π_l from $\mathcal{A}$ (Definition 6), the action node representing a at level i would have at least one (propositional or numerical) precondition node that is unsupported.

If conditions (2) or (3) held, then we can use a similar argument to show that the special last action node a_{end} would have at least one unsupported precondition node. Thus, π_l must be a valid plan for Π, and hence by Lemma 1 also the parallel plan of $\mathcal{A}$ must be valid for Π. □

4. Local search for NA-graphs

In this section, we present some search techniques for solving numerical planning problems through NA-graphs. We start with some preliminaries and an overview of the search process, which is similar to the method proposed in our earlier work on action graphs for propositional and temporal planning [15], extended with the management of the numerical information in the new plan representation. Then, we present the basic search neighborhood for NA-graphs and new heuristic techniques for (a) generating a solution NA-graph efficiently, (b) restricting the size of the neighborhood to make the search more effective, and (c) computing a solution NA-graph representing a good quality plan.

4.1. Preliminaries and overview of the search process

In our approach, we search a space of (partial) plans where each search state is an NA-graph for the numerical planning problem Π under consideration. Starting from an initial NA-graph for Π (a partial plan for Π), a local search process transforms it into a solution NA-graph for Π through the iterative application of some graph modifications (assuming that Π is solvable). The two basic modifications of an NA-graph consist of extending the NA-graph to include a new action node, and reducing the NA-graph to remove an action node (and the relevant edges). When we add an action node to a level l of the NA-graph, the graph is extended by one level, all action nodes from l are shifted forward by one level (i.e., they are moved to their next level), and the new action is inserted at level l. Similarly, when we remove an action node from level l, the graph is shrunk by removing level l.

The addition/removal of an action node a at a level of the NA-graph may require a revision of the world states identified with the next levels of the graph that are influenced by a. Let $\mathcal{A}'$ be the NA-graph obtained by adding (or

removing) an action node at level l of the current NA-graph $\mathcal{A}$. Basically, the algorithm for updating the state $\langle I'_m, \bar{x}'_m \rangle$ identified with a level $m > l$ of $\mathcal{A}'$ performs a forward state propagation, starting from the world state $\langle I'_l, \bar{x}'_l \rangle$ identified with level l of $\mathcal{A}'$ (which is the same as the state identified with level l of $\mathcal{A}$), and updating level by level the world states associated with the levels of $\mathcal{A}'$ from level $l+1$ up to the last level of the graph. Each world state $\langle I'_m, \bar{x}'_m \rangle$ is obtained by applying the effects of the action a_{m-1} at level $m-1$ to world state $\langle I'_{m-1}, \bar{x}'_{m-1} \rangle$. More formally,

$$I'_m = \begin{cases} I_m & \text{if } m \leqslant l \\ I'_{m-1} \cup PE^+(a_{m-1}) - PE^-(a_{m-1}) & \text{otherwise} \end{cases}$$

$$\bar{x}'_m = \begin{cases} \bar{x}_m & \text{if } m \leqslant l \\ (\xi_1^{a_{m-1}} \circ \cdots \circ \xi_{e_{m-1}}^{a_{m-1}})(\bar{x}'_{m-1}) & \text{otherwise} \end{cases}$$

where $\bar{x}_m$ is the vector of values for the numerical variables in the world state identified with level m of $\mathcal{A}$, e_{m-1} is the number of numerical effects of action a_{m-1}, and $\xi_j^{a_{m-1}}$ is the j-th numerical effect of action a_{m-1}.

Before starting the search for a solution NA-graph, we construct an initial NA-graph. Two alternative possible initial graphs, that can easily be computed, are the empty graph (containing only the special actions a_{start} and a_{end}) and a graph representing a partial plan obtained by removing some actions from a plan given as input to the search. The second option is used for the plan optimization phase described in Section 4.5.

At each search step, we identify the *neighborhood* $N(\mathcal{A})$ of the current NA-graph $\mathcal{A}$, which is the set of the NA-graphs formed by the graphs obtained from $\mathcal{A}$ by applying a graph modification for repairing the *earliest* flawed level l of $\mathcal{A}$.[4] $N(\mathcal{A})$ will be formally defined in Sections 4.2 and 4.4. Each element in $N(\mathcal{A})$ is weighted according to a *heuristic evaluation function* estimating its quality. The NA-graph in $N(\mathcal{A})$ with the lowest weight is selected as the best candidate for being the next NA-graph in the search process (i.e., the next search state). Informally, the quality of an NA-graph $\mathcal{A}' \in N(\mathcal{A})$ is the number of search steps required to reach a solution NA-graph from $\mathcal{A}'$ (i.e., the *search cost* of $\mathcal{A}'$), combined with the additional *execution cost* of $\mathcal{A}'$, that is determined by the cost of the actions that will be added to $\mathcal{A}'$ (the cost of an action is determined by the problem objective function and the world state where the action is applied).

The basic search procedure for searching the space of NA-graphs is Walkplan [19], a randomized method similar to the Walksat procedure for satisfiability checking [44], but with some important differences in the initialization phase and in the definition and evaluation of the search neighborhood. According to Walkplan, the best element in $N(\mathcal{A})$ is the NA-graph which has the *lowest decrease of quality* with respect to $\mathcal{A}$. In order to escape from possible local minima, Walkplan uses a *noise parameter* p randomizing the search. Specifically, if there is an element of $N(\mathcal{A})$ that does not decrease the quality of $\mathcal{A}$, then this NA-graph is chosen as the next search state; otherwise, with probability p one of the graphs in $N(\mathcal{A})$ is randomly chosen, and with probability $1-p$ the next NA-graph is chosen according to the minimum value of the heuristic evaluation function. In addition, we use a short *tabu list* [22] (with a dynamic length initially set to 5 and automatically increased/decreased when the search reaches/escapes from a local minimum). If a user defined number of search steps is reached without finding a solution NA-graph, then the current NA-graph is re-initialized with the empty plan, and a new search is performed. This is repeated for at most a user defined maximum number of times.

If the numerical planning problem to solve includes a plan metric expression to optimize (a multi-criteria objective function), the search process can be repeated in order to compute alternative plans with better quality. This method generates a sequence of valid plans, each of which improves the quality of the previous ones. In order to guarantee that each plan in the sequence improves the quality of the previous plans, each time we repeat the search, we first run a graph initialization algorithm, called InitializeGraph, that modifies the planning problem by adding an extra numerical goal. This goal imposes that the value of the plan metric function for the new plan will be lower than the value of the plan metric function for the last generated plan (and thus also for all plans generated so far). Moreover, InitializeGraph computes the initial NA-graph of each new search by performing some changes to the last solution NA-graph that has been generated, using a particular strategy aimed at facilitating plan optimization. InitializeGraph is formally described in Section 4.5, where we focus on plan optimization for planning with numerical information.

[4] We have designed some alternative flaw selection strategies that are described and experimentally evaluated in [16]. We observed that the strategy preferring the earliest flawed level of the graph tends to perform better than the other strategies, and so in our planner it is used as the default strategy.

In order to give a more compact treatment and presentation of our techniques in the next sections, without loss of generality, we can normalize every numerical precondition to a relation of type either $>$ or $\geqslant$. Let $\psi_f(\bar{x}) = \top$, if $f(\bar{x})\{\geqslant, >\}\ 0$, $\bot$ otherwise, be the normalized version of a numerical precondition, we call function f the *normal function* of ψ_f.

4.2. Basic search neighborhood for NA-graphs

The definition of search neighborhood for an NA-graph $\mathcal{A}$ is based on the notions of *helpful* action node and *harmful* action node for a flawed level of $\mathcal{A}$. Intuitively, a node is a helpful (harmful) action node for a level l of $\mathcal{A}$ if its addition (removal) helps to support a precondition node at level l. In the following formal definitions, the reader should keep in mind that, when an action node is added to an NA-graph, the graph is extended by one level, while when an action node is removed, the graph is shrunk by one level.

Definition 8 *(Helpful action node).* Let $\mathcal{A}$ be an NA-graph with a flawed level l, $\mathcal{A}'$ the NA-graph derived by *inserting* an action node a into $\mathcal{A}$ at a level $i \leqslant l$, $\langle I_l, \bar{x}_l\rangle$ and $\langle I'_{l+1}, \bar{x}'_{l+1}\rangle$ the world states identified with levels l and $l+1$ of $\mathcal{A}$ and $\mathcal{A}'$, respectively. Action node a is helpful for l in $\mathcal{A}$ iff

- at level l $\mathcal{A}$ has a propositional precondition node that is not supported, and that becomes supported at level $l+1$ of $\mathcal{A}'$, or
- $f(\bar{x}'_{l+1}) - f(\bar{x}_l) > 0$ holds, where f is the normal function of a numerical precondition represented by a precondition node at level l of $\mathcal{A}$ that is not supported.

Note that if a numerical precondition node is unsupported, then the value of the corresponding numerical precondition normal function is negative. Hence, condition $f(\bar{x}'_{l+1}) - f(\bar{x}_l) > 0$ in the second item of the previous definition implies that in the extended NA-graph the "degree of unsatisfaction" for the numerical precondition under consideration is reduced.

Example 3 *(Helpful Action Node).* In the bottom NA-graph of Fig. 2 ($\mathcal{A}_2$), action node a_{new} has a numerical effect increasing the value of x by 100. This action node is helpful for level 2 of the top-left NA-graph of Fig. 2 ($\mathcal{A}$) because: (i) level 2 of $\mathcal{A}$ contains the node labeled `x >= 100`, which is an unsupported numerical precondition node of action node $a2$ in $\mathcal{A}$; (ii) the numerical value of x at level 2 of $\mathcal{A}$ is 50, while at level 3 of the NA-graph $\mathcal{A}_2$ obtained by adding a_{new} at level 2 of $\mathcal{A}$, the value of x is 150; (iii) the normal function of `x >= 100` is $f(\bar{x}) = x - 100$; (iv) $f(\bar{x}'_3) - f(\bar{x}_2) > 0$ holds, where $\bar{x}'_3$ is the tuple of values of the numerical fluents in the world state identified with level 3 of $\mathcal{A}_2$. On the contrary, if a_{new} were added at level 1 of $\mathcal{A}$, a_{new} would *not* be helpful for level 2, because of effect `assign(x,50)` of $a1$.

It is important to note that, in order to support a numerical precondition node in an NA-graph, it can be necessary to add more than one action node to the graph. For example, if in the world state identified with a level l of the graph the value of a numerical fluent x is 0, then in order to support a numerical node labeled `x > 25` at level l using an action increasing the value of x by 10 units, we need three occurrences of such an action. We say that each of these three actions *contributes to support* the node labeled `x > 25`.

Definition 9 *(Harmful action node).* Let $\mathcal{A}$ be an NA-graph with a flawed level l, $\mathcal{A}'$ the NA-graph derived by *removing* an action node a at level $i \leqslant l$ from $\mathcal{A}$, $\langle I_l, \bar{x}_l\rangle$ and $\langle I'_{l-1}, \bar{x}'_{l-1}\rangle$ the world states identified with levels l and $l-1$ of $\mathcal{A}$ and $\mathcal{A}'$, respectively. Action node a is harmful for l in $\mathcal{A}$ iff

- $i = l$ (a precondition node of a is not supported in $\mathcal{A}$), or
- $i < l$ and a propositional precondition node that is not supported at level l of $\mathcal{A}$ becomes supported in $\mathcal{A}'$, or
- $i < l$ and $f(\bar{x}'_{l-1}) - f(\bar{x}_l) > 0$ hold, where f is the normal function of a numerical precondition represented by a precondition node at level l of $\mathcal{A}$ that is unsupported.

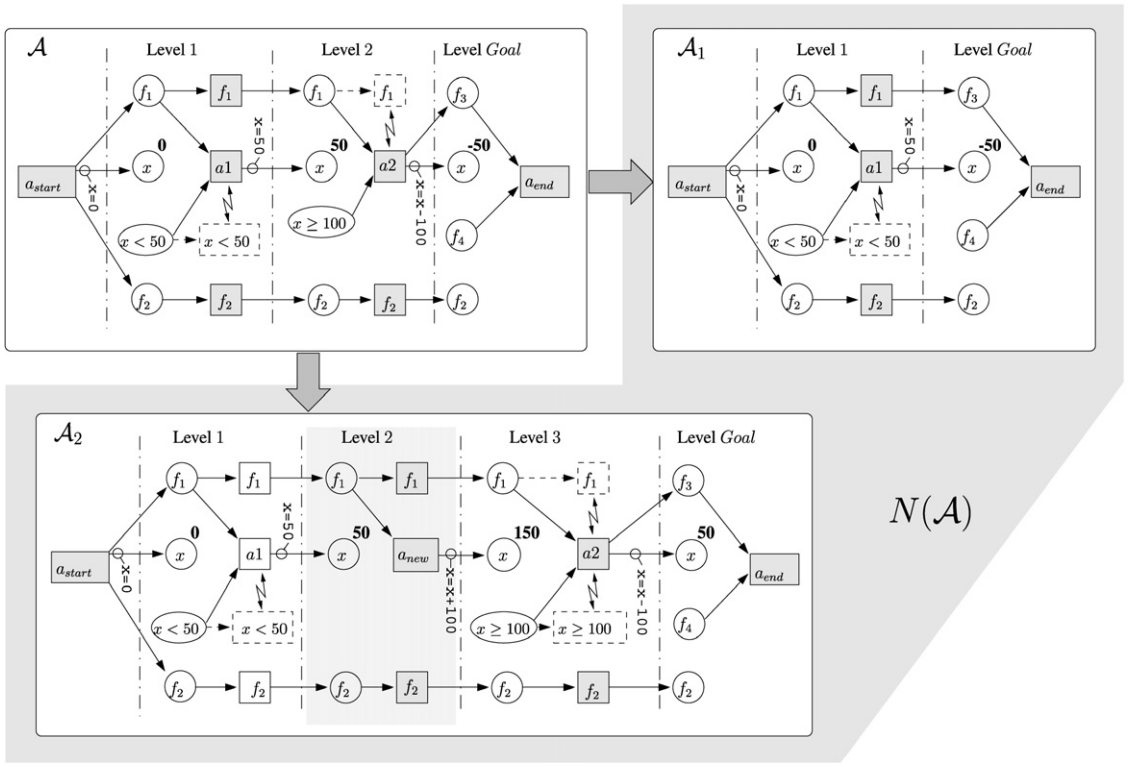


Fig. 2. Example of search neighborhood of a numerical action graph $\mathcal{A}$ for the flawed level 2, where numerical precondition $x \geqslant 100$ is unsupported. We assume that a_{new} is the only available action that increases the value of numerical fluent x. Square nodes are action nodes; circle nodes are propositional nodes; elliptical nodes are numerical fluent nodes. Dashed edges (in $\mathcal{A}_1$ and $\mathcal{A}_2$) form chains of no-ops that are blocked by a mutex relation.

Concerning the first condition in the definition of a harmful action node a, if $i = l$, since l is flawed, action a has a precondition node that is not supported. Thus, if we remove a from the current NA-graph $\mathcal{A}$, we also remove its unsupported precondition nodes from $\mathcal{A}$. Concerning the second and third conditions, intuitively, an action node is harmful when its *presence* in the graph prevents the satisfaction of a propositional precondition, or it degrades the (partial) satisfaction of a numerical precondition.

Example 4 *(Harmful Action Node).* Action node $a2$ at level 2 of $\mathcal{A}$ (top-left graph of Fig. 2) is harmful for level 2, because the graph $\mathcal{A}_1$ (top-right graph of Fig. 2) obtained by removing $a2$ from $\mathcal{A}$ does not contain the unsupported precondition node labeled $\mathtt{x} \geqslant \mathtt{100}$. On the contrary, action node $a1$ at level 1 of $\mathcal{A}$ is *not* harmful for level 2, because its removal would satisfy none of the three conditions in Definition 9.

Definition 10 *(Search neighborhood).* Let l be a flawed level in an NA-graph $\mathcal{A}$. The search neighborhood $N(\mathcal{A})$ of $\mathcal{A}$ for level l is the set of NA-graphs derived from $\mathcal{A}$ by adding an action node that is helpful for l or by removing an action node that is harmful for l.

In general, the previous definition could rule out graph modifications contributing to remove an unsupported numerical precondition in the flawed level under consideration. For example, if the normal function of the unsupported precondition is not monotonic in each fluent separately, the insertion of some non-helpful action nodes could contribute to support it. The simplest way to deal with these cases would be considering the addition or removal of *any* action node affecting the unsupported precondition under consideration, even if it is not helpful or harmful. However, this could add many useless elements to the search neighborhood, making the search process inefficient. Moreover, we have experimentally observed that for a large set of benchmark problems our definition of search neighborhood is effective and efficient.

Each NA-graph in the neighborhood $N(\mathcal{A})$ of an NA-graph $\mathcal{A}$ is evaluated by using a *heuristic evaluation function* E. Let $\mathcal{A}_a^{i/r}$ be the NA-graph in $N(\mathcal{A})$ obtained by either adding ($\mathcal{A}_a^i$) or removing ($\mathcal{A}_a^r$) action node a to/from $\mathcal{A}$ for repairing a flawed level l. In general, for numerical planning problems, function E is defined as the sum of two weighted terms

$$E(\mathcal{A}_a^{i/r}) = \alpha \cdot SearchCost(a)^{i/r} + \beta \cdot ExecCost(a)^{i/r}$$

where

- $SearchCost(a)^{i/r}$ is a heuristic estimate of the *search cost* of $\mathcal{A}_a^{i/r}$, i.e., an estimate of the minimum number of search steps required to repair the new flaws introduced in $\mathcal{A}_a^{i/r}$ and the flaws remaining in $\mathcal{A}_a^{i/r}$ at the level corresponding to l;
- $ExecCost(a)^{i/r}$ is a heuristic estimate of the *execution cost* of $\mathcal{A}_a^{i/r}$, i.e., an estimate of the total cost of the new actions defining the search cost of $\mathcal{A}_a^{i/r}$, where the cost of an action is defined by the plan metric function, as will be described in Section 4.5;
- coefficients α and β normalize the two terms of E to a value in [0, 1], and weight the relative importance of the terms.

If the numeric planning problem has no objective function, then the execution cost term is not present in the evaluation function E. The values of α and β are automatically and dynamically computed during search as described in [15].[5]

As will be discussed in Section 6.2, heuristic evaluation function E is very useful for the good performance of our approach. In the next subsection, we focus on the search cost evaluation in E, while in Section 4.5 we describe the execution cost evaluation.

4.3. Relaxed-plan heuristics for fast generation of solution NA-graphs

$SearchCost(a)^i$ is estimated by constructing a *relaxed* numerical plan π_{R}. Plan π_{R} is a relaxed solution of the numerical problem in which the initial state is the world state identified with the level i of $\mathcal{A}$ where we are considering inserting a for repairing the (earliest) flawed graph level l, and the goals are the propositional and numerical fluents represented by:

- the unsupported precondition nodes of a, denoted by $Pre(a)$,
- the unsupported precondition nodes at level l that would remain unsupported after adding a to $\mathcal{A}$, denoted by $Unsup(l)$, and
- the supported precondition nodes of other action nodes in $\mathcal{A}$ that would *become un*supported by adding a to $\mathcal{A}$, denoted by $Threats(a)$.

Plan π_{R} is constructed in two phases. First, we compute the relaxed subplan of π_{R} for achieving $Pre(a)$ using the state $\langle I_i, \bar{x}_i \rangle$ identified with the level i of $\mathcal{A}$ as the initial world state. Then, we compute the subplan of π_{R} for achieving $Unsup(l)$ and $Threats(a)$, using the state $\langle I_i, \bar{x}_i \rangle$ modified by applying the action labeling a to it as the initial world state. The second subplan can reuse any action in the first computed relaxed subplan.

The evaluation of an NA-graph of $SearchCost(a)^r$ is similar. In this case, the goals of the relaxed plan are the propositional and numerical fluents represented by

- the precondition nodes supported by a (at any level of the graph) that would become *un*supported by removing a, denoted by $UnsupFacts(a)$, and
- when a is at a level $i < l$, the unsupported precondition nodes at level l that do not become supported by removing a, denoted by $Unsup(l)$.

[5] We empirically observed that, for a large set of benchmark problems, the dynamic setting of α and β is useful in some cases, but in general it does not significantly affect the performance of our planner.

The goals of the second point are considered only if $i < l$ because, if $i = l$, then by removing a all flaws at l are eliminated, while, if $i < l$, the removal of a could leave some flaws at level l.

In the following, first we present our algorithm for computing relaxed plans for numerical planning, and then we give a detailed description on how the terms of function E are defined using relaxed plans.

4.3.1. Computation of relaxed plans for numerical planning problems

The plan π_{R} constructed for evaluating the search cost of an NA-graph in the search neighborhood is relaxed in the sense that it does not consider the possible interference between actions in π_{R}, and the numerical preconditions of the actions in π_{R} are evaluated in a relaxed way, using estimated minimum and maximum (min/max) values for the numerical fluents involved in the plan.[6] The estimated min/max values of the numerical fluents involved in π_{R} are *monotonically* updated by using the numerical effect functions of the actions in π_{R}. Specifically, if a numerical effect function ξ of an action in π_{R} increases the value of x, then we increase the max value of x; while, if ξ decreases the value of x, then we decrease the min value of x in π_{R}. In other words, the max value of x is not affected by the effects decreasing x, and can never decrease. Analogously, for the min value of x, which can never increase. The initial min/max values of the numerical fluents is defined by the initial world state for π_{R}.

Let x be a numerical variable involved in the normal function f of a numerical precondition ψ_f (written by distributing the multiplications/divisions over additions/subtractions). If x appears in a monotonically increasing (w.r.t. x) term of f multiplied by a positive coefficient or in a monotonically decreasing (w.r.t. x) term of f multiplied by a negative coefficient, then for this term the *relaxed* evaluation of ψ_f considers the estimated max value of x in π_{R}; otherwise, the evaluation of ψ_f considers the estimated min value of x in π_{R}. Similar conditions are used to consider either the estimated max or min value for each fluent y involved in a numerical effect function, when we use this effect to update the min/max values of a numerical fluent. Under the assumption that the normal function of each numerical precondition and each numerical effect function are monotonic in every fluent separately (for the reachable fluent values), this method for computing and using the fluent min/max values ensures that in the relaxed evaluation of ψ_f we always consider an *upper bound* of the value of f.

For example, let $f(x, y) = x - y - 10$ be the normal function of numerical precondition $\psi_f = \top$, if $x - y > 10$, $\bot$ otherwise. The evaluation of ψ_f in the relaxed plan π_{R} considers the max value of x and the min value of y (which are determined by the effects of the actions currently in π_{R} as described above). If the (current) max value of x is 10 and the (current) min value of y is 5, then, according to these values, ψ_f is not satisfied by the relaxed evaluation, because the (estimated) max value of f is -5.

Clearly, our computation of the fluent min/max values is an approximation of their exact values, and the relaxed evaluation of the numerical preconditions is an optimistic estimation of their truth. On the other hand, the relaxed evaluation makes the numerical preconditions much easier to achieve for the relaxed plan. This is very important for the effectiveness of the overall search process, because many relaxed plans can be computed during search (several of them at each search step), and an exact method for achieving the numerical preconditions would make the search neighborhood evaluation a severe bottleneck for the search process.

Fig. 3 gives a formal description of our algorithm, called RelaxedNumPlan, for computing a relaxed numerical plan. Given a set of goals G, an initial world state $s = \langle I, \bar{x} \rangle$, and a possibly empty bag of actions A, RelaxedNumPlan recursively computes a bag of actions (*Rplan*) forming a relaxed plan π_{R} achieving G from s.[7]

Plan π_{R} is constructed by a backward process that can reuse the input actions in A for satisfying the action preconditions or the goals. Initially, the set of goals G is reduced by the set of initial facts I, the set of achieved facts F is set to I, and the bag of actions in the current relaxed plan (*Acts*) is set to A (step 1). Step 3 of the algorithm updates the estimated min/max values of the numerical fluents, while steps 4–5 augment set F with the propositional positive effects and the satisfied numerical preconditions of the actions in *Acts*. Then step 7 selects an action b for achieving a precondition or goal q in $G - F$ (*BestAction*(q)), and steps 8–9 derive a bag of actions forming a relaxed subplan for the unsatisfied preconditions of b. Finally, step 10 returns the computed relaxed plan.

[6] An estimation of the min/max values of the numerical fluents is also computed in the heuristics developed for SAPA$^{\mathcal{M}ps}$ [2] and Metric-FF [27]. More precisely, Metric-FF only computes upper bounds after conversion of the *numeric tasks* of the planning problem into *linear normal form*.

[7] A bag of actions is a collection of actions in which we can have multiple copies of the same action.

RelaxedNumPlan($G, \langle I, \bar{x}\rangle, A$)
Input: A set of goals G, an initial world state $\langle I, \bar{x}\rangle$, and a bag of actions A;
Output: An estimated minimal bag of actions *Acts* forming a relaxed plan for achieving G from $\langle I, \bar{x}\rangle$.

1. $G \leftarrow G - I$; $F \leftarrow I$; $Acts \leftarrow A$;
2. **forall** $q \in G$ **do**
3. $\quad \langle \bar{x}_{min}, \bar{x}_{max}\rangle \leftarrow$ ComputeMinMax($\bar{x}, \bigcup_{a_i \in Acts} NE(a_i)$);
4. $\quad F \leftarrow \bigcup_{a \in Acts} PE(a)^+$;
5. $\quad F \leftarrow F \cup \{q \in G \cup \bigcup_{a \in Acts} NP(a) \mid q$ is satisfied using $\bar{x}_{min}$ and $\bar{x}_{max}\}$;
6. $\quad$ **if** $q \notin F$ **then**
7. $\quad\quad b \leftarrow BestAction(q)$;
8. $\quad\quad Rplan \leftarrow$ RelaxedNumPlan($PP(b) \cup NP(b), \langle I, \bar{x}\rangle, Acts$);
9. $\quad\quad Acts \leftarrow Rplan + Occurrences(b, q)$;
10. **return** *Acts*.

Fig. 3. An algorithm for computing a relaxed plan for a set of goals G from a world state $\langle I, \bar{x}\rangle$ reusing the actions in (possibly empty) bag A. ComputeMinMax monotonically updates $\bar{x}_{min}$ and $\bar{x}_{max}$ (i.e., the min and max values of the numerical fluents in the planning problem) as described in the text; "+" is the operator extending a bag of actions with one bag of actions; $Occurrences(b, q)$ is a bag of actions computed as described in the text.

We now describe in detail $BestAction(q)$. The action chosen for achieving q is an action b minimizing the sum of

(I) the maximum of the heuristic number of actions required to support each precondition p of b from $\langle I, \bar{x}\rangle$, denoted by $Num\_acts(p, \langle I, \bar{x}\rangle)$;
(II) the estimated number of supported precondition nodes in $\mathcal{A}$ that would become unsupported by adding b to $\mathcal{A}$ ($Threats(b)$);
(III) the minimal bag of occurrences of b required to achieve q, denoted by $Occurrences(b, q)$.

More formally, $BestAction(q)$ at step 7 of RelaxedNumPlan is an action satisfying

$$\underset{\{b \in A_q\}}{\text{ARGMIN}}\Big\{\Big(\max_{p \in PP(b) \cup NP(b) - F} Num\_acts(p, \langle I, \bar{x}\rangle)\Big) + \big|Threats(b)\big| + \big|Occurrences(b, q)\big|\Big\}$$

where: F is the set computed by steps 1, 4–5 of RelaxedNumPlan formed by the (positive) effects of the actions currently in *Acts* (i.e., the set of actions in the relaxed plan π_{R} under construction), and the numerical preconditions satisfied in the portion of π_{R} computed so far; A_q is the set of the actions with propositional effect q, or with an effect increasing the output value of the normal function of q in π_{R} (if q is a numerical condition).

The term $Num\_acts(p, \langle I, \bar{x}\rangle)$ in the definition of $BestAction(q)$ is computed by *reachability analysis* using a polynomial algorithm similar to the one proposed in [15].

In order to efficiently compute $Threats(b)$ in the definition of $BestAction(q)$, we use an approximation of the actual threats that action b would generate if it were added to the current NA-graph $\mathcal{A}$ at level i (the level where we are considering the addition of the new action a for repairing or contributing to repair the selected flawed level l of $\mathcal{A}$). The approximation is given by the fact that, in order to determine whether a numerical precondition of an action at a level $k \geqslant i$ would become unsupported by adding b at level $i \leqslant l$, we evaluate the precondition in world state $\langle I_i, \bar{x}_i\rangle$ of $\mathcal{A}$, rather than in world state $\langle I_{k+1}, \bar{x}_{k+1}\rangle$ of $\mathcal{A}$ extended with a (i.e., we ignore the actions from level $i+1$ to level k that could affect the value of x). While an exact evaluation could make $BestAction(q)$ more accurate, it would require propagating the effects of b through $\mathcal{A}$ from level i to level k. But this propagation has to be done multiple times during the construction of a relaxed plan, and since the graph can have many levels, the overall overhead can make the computation of the neighborhood evaluation function too slow.

The term $Occurrences(b, q)$ in the definition of $BestAction(q)$ and step 9 of RelaxedNumPlan is the minimal bag of occurrences of action b that is required to satisfy precondition q according to the current values of $\bar{x}_{min}$ and $\bar{x}_{max}$. The general idea is that, if b is selected for supporting q in the relaxed plan under construction, then additional actions could be required to fully support q, and we estimate a minimal bag of such actions. $Occurrences(b, q)$ is computed

EvalAdd(l, a)
Input: A flawed level l in the current NA-graph $\mathcal{A}$; a helpful action node a for level l.
Output: A bag of actions (*Rplan*) forming a relaxed plan.

1. $g \leftarrow$ the precondition node at level l of $\mathcal{A}$ for which a is helpful;
2. $i \leftarrow$ level of $\mathcal{A}$ where we are evaluating the addition of a;
3. *Rplan* $\leftarrow$ RelaxedNumPlan($Pre(a), \langle I_i, \bar{x}_i \rangle, \emptyset$); / * *relaxed subplan for* $Pre(a)$ * /
4. $I_i^* \leftarrow I_i \cup PE(a)^+ - Threats(a)$; $\bar{x}_i^* \leftarrow \xi_1^a \circ \cdots \circ \xi_k^a(\bar{x}_i)$;
5. *Rplan* $\leftarrow$ RelaxedNumPlan($Unsup(l) \cup Threats(a), \langle I_i^*, \bar{x}_i^* \rangle, Rplan$);
   /* *relaxed subplan for* $Unsup(l)$ and $Threats(a)$ * /
6. Extend *Rplan* with the minimal bag of occurrences of a for satisfying the action precondition represented by g from $\langle I_i, \bar{x}_i \rangle$.
7. **return** *Rplan*.

EvalDel(l, a)
Input: A flawed level l in $\mathcal{A}$ and a harmful action node a at a level $i \leqslant l$.
Output: A bag of actions *Rplan* forming a relaxed plan.

1. *Rplan* $\leftarrow$ RelaxedNumPlan($UnsupFacts(a), \langle I_i, \bar{x}_i \rangle, \emptyset$);
2. **if** $i < l$ **then** *Rplan* $\leftarrow$ RelaxedNumPlan($Unsup(l), \langle I_i, \bar{x}_i \rangle, Rplan$);
3. **return** *Rplan*.

Fig. 4. Algorithms for computing the search cost of adding (removing) a helpful (harmful) action node a at a level $i \leqslant l$ of an NA-graph for repairing a flawed level l. $PE(a)^+$ is the set of positive effects of action a, and $\xi_1^a, \ldots, \xi_k^a$ are the k numerical effect functions of action a.

by repeatedly applying the effects of b to the current values of $\bar{x}_{min}$ and $\bar{x}_{max}$, which (temporarily) changes their values as described above, until q is satisfied by $\bar{x}_{min}$ and $\bar{x}_{max}$.[8]

4.3.2. Use of relaxed plans in the neighborhood evaluation function

As noted at the beginning of Section 4.3, the search cost terms of the heuristic evaluation function are computed using relaxed numerical plans. More precisely, we have

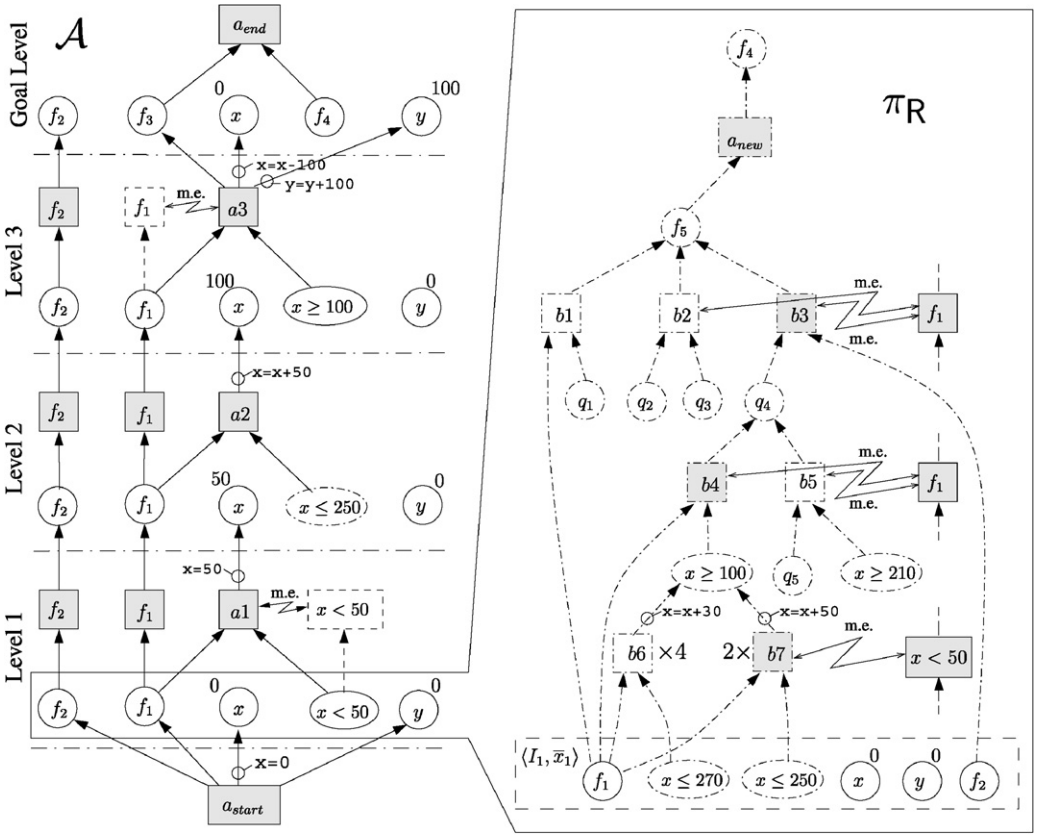
$$SearchCost(a)^{i/r} = \left|\pi_{\mathsf{R}}^{i/r}\right| + \sum_{a' \in \pi_{\mathsf{R}}^{i/r}} \left|Threats(a')\right|$$

where: $\pi_{\mathsf{R}}^{i/r}$ are the bags of actions forming the relaxed plans computed for estimating the search cost of the NA-graph obtained by adding/removing action node a; $Threats(a')$ are the supported precondition nodes in the current NA-graph $\mathcal{A}$ that would become unsupported by adding an action node representing a' at the level of $\mathcal{A}$ under consideration for a.

For the sake of simplicity, in the following we will use the same notation to indicate an action node and the action represented by an action node. The exact meaning will be clear from the context.

Fig. 4 describes EvalAdd and EvalDel, the two high-level algorithms using RelaxedNumPlan for computing π_{R}^i and π_{R}^r, respectively. Concerning EvalAdd, step 3 uses algorithm RelaxedNumPlan for computing a bag of actions (*Rplan*) forming a relaxed numerical subplan achieving the preconditions of the input action a ($Pre(a)$) from the world state $\langle I_i, \bar{x}_i \rangle$ identified with level i of the current NA-graph where we are considering adding action node a. Step 4 computes the new world state obtained by applying the effects of a to state $\langle I_i, \bar{x}_i \rangle$. Step 5 uses RelaxedNumPlan for extending *Rplan* with the bag of actions forming a relaxed subplan for $Unsup(l)$ and $Threats(a)$, possibly reusing the actions already in *Rplan*. Finally, step 6 extends *Rplan* with the minimal bag of occurrences of a that are required to satisfy the (propositional or numerical) preconditions represented by a precondition node at the flawed level l for which a is helpful.

[8] For the special case in which the effects of b set a numerical fluent in q to a particular *constant* value, $Occurrences(b, q) = [b] + O$, where O is the minimal bag of occurrences of *any* action c in the numerical planning problems that is required to satisfy q according to the current values of $\bar{x}_{min}$ and $\bar{x}_{max}$.



a_1 : refuel(A,L1)	a_{new} : debark(P,A,L1)	q_1 : at(P,L1)
a_2: refill-100(A,L1)	$b1$: board(P,A,L1)	q_2 : at(A,L3)
a_3: fly(A,L1,L2)	$b2$: board(P,A,L3)	q_3 : at(P,L3)
f_1: at(A,L1)	$b3$: board(P,A,L2)	q_4 : at(A,L2)
f_2: at(P,L2)	$b4$: fly(A,L1,L2)	q_5 : at(A,L3)
f_3: at(A,L2)	$b5$: fly(A,L3,L2)	x : fuel(A)
f_4: at(P,L1)	$b6$: refill-30(A,L1)	y : fuel_used
f_5: in(P,A)	$b7$: refill-50(A,L1)	

Precond.	*Num_acts*
q_1	5
q_2	3
q_3	5
q_4	3
q_5	3
$x \geq 100$	2
$x \geq 210$	5

Fig. 5. An example of relaxed numerical plan π_R constructed by RelaxedNumPlan. Square nodes are action nodes; circle nodes are either fact nodes or fluent nodes; elliptical nodes are numerical precondition nodes; solid nodes are nodes in the current NA-graph $\mathcal{A}$ extended with a_{new}; dotted nodes are graph nodes considered during the construction of π_R; gray dotted nodes are nodes selected by RELAXEDNUMPLAN. The "×" on the side of an action node indicates the corresponding number of occurrences. The assignment statements on the edge labels abbreviate the corresponding PDDL assignment operators.

Similarly, EvalDel uses RelaxedNumPlan with input goals *UnsupFacts*(a) (step 1) and, if the level of the removed action node a precedes the flawed level under consideration ($i < l$), also with goals *Unsup*(l) (step 2).

4.3.3. An example of computing and using relaxed numerical plans

In Fig. 5, we give an example of relaxed plan π_R in the context of the ZenoTravel domain. This relaxed plan is used to evaluate the addition of action node `debark(P,A,L1)` (a_{new}) at level 1 of the NA-graph $\mathcal{A}$ on the left side of the figure, which is helpful for the goal level, because its insertion supports precondition node `at(P,L1)` (f_4). The figure has two tables. The table on the left side is used to abbreviate fact and action names; the table on the right side gives the *Num_ acts*-values of the relevant facts, which are computed by reachability analysis.

In the first run of RelaxedNumPlan (step 3 of EvalAdd), the goal of π_R is the unsatisfied precondition `in(P,A)` (f_5) of a_{new}; the initial world state, $\langle I_1, \bar{x}_1 \rangle$, is defined by the propositional fact nodes supported at level 0 of $\mathcal{A}$, i.e.,

`at(A,L1)` (f_1) and `at(P,L2)` (f_2), and by the values associated with the numerical fluent nodes `fuel(A)` (x) and `fuel_used` (y) at level 0. The initial min/max value of x and y is 0 (i.e., the value of x and y in state $\langle I_1, \bar{x}_1\rangle$). Suppose that the only available actions achieving f_5 are `board(P,A,L1)` ($b1$), `board(P,A,L3)` ($b2$) and `board(P,A,L2)` ($b3$), and thus that the set of actions A_{f_5} examined by *BestAction*(f_5) at step 7 of RelaxedNumPlan is $\{b1, b2, b3\}$. The value of the first term in the argmin expression defining *BestAction* is 5 for $b1$, 5 for $b2$ and 3 for $b3$; the value of the second term is 0 for $b1$, and 1 for both $b2$ and $b3$; finally, the value of the third term is 1 for both $b1$, $b2$ and $b3$. It follows that *BestAction*(f_5) returns $b3$, and step 9 will add one occurrence of $b3$ to the current bag of actions forming π_R (*Rplan*).

In the recursive call of RelaxedNumPlan for achieving the preconditions of $b3$ (step 8), step 1 removes `at(P,L2)` (f_2) from the new goal set G (i.e., the preconditions of $b3$), because f_2 is already satisfied in the current relaxed plan, and so the only unsatisfied precondition in G is `at(A,L2)` (q_4). Suppose that the only available actions achieving q_4 are `fly(A,L1,L2)` ($b4$) and `fly(A,L3,L2)` ($b5$). Each of these actions is evaluated, and step 7 selects $b4$.

In the recursive call of RelaxedNumPlan for achieving the preconditions of $b4$, precondition `fuel(A)` $\geqslant 100$ ($x \geqslant 100$) is unsatisfied, because the current max value of x (i.e., 0) is not greater than or equal to 100. Suppose that the only available actions affecting the value of x are `refill-30(A,L1)` ($b6$) and `refill-50(A,L1)` ($b7$). Then, the set of actions $A_{x \geqslant 100}$ examined by *BestAction*($x \geqslant 100$) at step 7 is $\{b6, b7\}$. The preconditions of $b6$ and $b7$ are already satisfied in state $\langle I_1, \bar{x}_1\rangle$, and so the value of the first term in the definition of *BestAction*($x \geqslant 100$) is zero for both $b6$ and $b7$. The value of the second term is 0 for $b6$ (because *Threats*($b6$) $= \emptyset$) and 1 for $b7$ (because *Threats*($b7$) $= \{x < 50\}$). Concerning the third term in the definition of *BestAction*($x \geqslant 100$), we have:

- $|Occurrences(b6, x \geqslant 100)| = |[b6, b6, b6, b6]| = 4$, since, in order to satisfy $x \geqslant 100$ with action $b6$, according to the current max value of x (0), we need 4 occurrences of $b6$;
- $|Occurrences(b4, x \geqslant 100)| = |[b7, b7]| = 2$, since, in order to satisfy $x \geqslant 100$ with $b7$, according to the current max value of x (0), we need 2 occurrences of $b7$.

Thus, *BestAction*($x \geqslant 100$) is $b7$, and step 9 will add two occurrences of $b7$ to the current bag of actions forming π_R (*Rplan*). The insertion of such a pair of actions in the relaxed plan will increase the max value of x by 100 units, while its min value will remain unchanged.

The next recursive call of RelaxedNumPlan is applied to the precondition of $b7$ (f_1 and $x \leqslant 250$). Both conditions are satisfied with respect to $\langle I_1, \bar{x}_1\rangle$ and $\bar{x}_{min}/\bar{x}_{max}$; $f_1 \in I_1$, while the current min value of x (i.e., 0) satisfies $x \leqslant 250$. (This condition is evaluated using $\bar{x}_{min}$ because in the normalized form of the condition x has a negative coefficient.)

Summarizing, the fourth recursive call of RelaxedNumPlan (for the preconditions of $b7$) returns the empty bag; the third recursive call (for the preconditions of $b4$) returns the bag $[b7, b7]$; the second one (satisfying the preconditions of $b3$) returns the bag $[b4, b7, b7]$; finally, the first (top-level) run of RelaxedNumPlan returns the bag $[b3, b4, b7, b7]$.

Concerning the second run of RelaxedNumPlan (step 5 of EvalAdd), we assume that at goal level of $\mathcal{A}$ (the flawed level under consideration) there is no other flaw, and that the insertion of a_{new} at level 1 threats no supported precondition of $\mathcal{A}$ (i.e., *Threats*(a_{new}) $= \emptyset$). Therefore, the second run of RelaxedNumPlan adds no action to the plan computed so far.

In addition to the bag of actions forming the relaxed plans for *Pre*(a_{new}), step 6 of EvalAdd adds one occurrence of a_{new}, yielding a relaxed plan with 5 actions. Finally, since in the relaxed plan there are three actions ($b3$, $b4$ and $b7$) threatening some supported precondition nodes in $\mathcal{A}$ (f_1 and $x < 50$), the search cost of the NA-graph obtained by adding a_{new} to level 0 of the current NA-graph is estimated to be 8.

4.4. Heuristic search neighborhood

In a local search procedure, the effectiveness of the heuristic function evaluating the elements of the search neighborhood can significantly be affected by the size of the neighborhood. If this is very large, the neighborhood evaluation could be too slow, making the whole search procedure ineffective. In the basic search neighborhood defined in Section 4.2, we have, in the worst case, an NA-graph for each helpful/harmful action node at each level of the graph. Hence the neighborhood size is $O(A_l \cdot L)$, where A_l is the number of domain actions affecting the flawed level l under consideration and L is the number of levels in the current NA-graph.

Table 1
Minimum, mean and maximum size of the search neighborhood (N) and the heuristic search neighborhood (HN) for the numerical version of some benchmark problems with small, medium and large sizes

Problems	size of N			size of HN		
	Min	Mean	Max	Min	Mean	Max
Depots-1	2	10.4	36	1	3.61	10
DriverLog-1	3	5.60	12	3	3.62	5
Openstacks-1	6	24.7	142	1	2.00	3
Pathways-1	2	3.68	19	2	2.35	4
Rovers-1	2	12.9	33	2	3.00	5
Satellite-1	2	6.39	13	1	5.89	16
Settlers-1	1	33.5	1026	1	6.34	46
TPP-1	5	11.1	35	5	6.87	10
UMTS-10	2	3.61	8	1	1.77	2
ZenoTravel-1	4	4.00	4	3	3.66	5
Depots-10	6	78.4	626	1	6.7	34
DriverLog-10	3	36.6	116	2	5.23	11
Openstacks-10	1	34.9	543	1	1.99	3
Pathways-10	1	69.6	4170	1	3.34	15
Rovers-10	2	179	752	1	5.15	12
Satellite-10	1	94.5	781	2	10.5	18
Settlers-10	1	74.8	3977	1	9.36	73
TPP-10	1	42.9	182	1	10.10	14
UMTS-50	2	28.8	170	1	1.80	2
ZenoTravel-10	1	69.8	521	2	5.8	9
Depots-20	8	1053	16,534	1	19	222
DriverLog-20	2	1391	20,701	2	21.5	81
Openstacks-20	1	296	1802	1	2.86	52
Pathways-20	1	88.3	2906	1	3.38	29
Rovers-40	2	5073	47,736	1	11.3	127
Satellite-20	1	380.8	8181	1	25.65	299
Settlers-20	1	144	8869	1	14.9	136
TPP-30	1	214	3360	1	24.4	46
ZenoTravel-20	1	2357	57,214	2	18.79	43

We empirically observed that, for many search steps, the size of the basic search neighborhood is significantly large. Table 1 shows the minimum, mean and maximum size of the neighborhood for some known benchmark problems. For the small-size problems the neighborhood is relatively small, except in some cases for the Openstacks and Settlers problems. However, for the medium-size problems the average size is significantly greater, and for the large-size problems in many cases it has a prohibitive size (i.e., as confirmed by the experiments in Section 6.2, evaluating these neighborhoods is computationally very expensive).

Here we propose an alternative restricted search neighborhood, which we call the *Heuristic Search Neighborhood* (*HN*). While having a smaller neighborhood makes its evaluation faster, clearly not every restriction can speed up the overall search process. In principle, we would like to leave out only the "worst elements" (i.e., those NA-graphs requiring the search to perform more steps). The elements of the restricted neighborhood are obtained by a *pre-evaluation* of the neighborhood elements using a fast heuristic function. This function is the same as the one that we developed for selecting the actions forming a numerical relaxed plan. Specifically, let $\mathcal{A}_a^i$ be the NA-graph obtained by adding an action node a to a level l_a of $\mathcal{A}$ that achieves precondition q in $\mathcal{A}$, we define the quality $H(\mathcal{A}_a^i)$ of $\mathcal{A}_a^i$ as

$$H(\mathcal{A}_a^i) = \Big(\max_{p \in PP(a') \cup NP(a')} Num\_acts\big(p, \langle I_{l_a}, \bar{x}_{l_a}\rangle\big)\Big) + \big|Threats(a')\big| + \big|Occurrences(a', q)\big|,$$

where $Num\_acts(p, \langle I_{l_a}, \bar{x}_{l_a}\rangle)$ is equal to zero if p holds in the world state associated with level l of $\mathcal{A}_a^i$. The heuristic search neighborhood $HN(\mathcal{A})$ is restricted because it considers only a subset of the NA-graphs derived by adding a helpful node: *for each action a, instead of containing all the NA-graphs obtained by adding a to any level* $l' \leqslant l$ *where a is helpful for l,* $HN(\mathcal{A})$ *only contains the NA-graph obtained by adding a to the* best level *of the graph, where the*

best level is estimated by minimizing H over all NA-graphs $\mathcal{A}_a^i$ resulting from adding a to a level $l' \leqslant l$. Like the basic search neighborhood, $HN(\mathcal{A})$ also contains all the NA-graphs derived by removing the harmful actions.

Clearly, computing function H is faster than computing function E, but H is less accurate than E, and it could be the case that the use of $HN(\mathcal{A})$ leads to pruning some elements that according to E have a better quality (number of search steps required to derive a solution NA-graph). On the other hand, the neighborhood size is significantly reduced (in the worst case, it becomes $O(A_l + L)$), which makes each step of the local search procedure faster.

Table 1 compares the sizes of the basic neighborhood N and of the heuristic neighborhood HN. The gray data in the mean and max columns concern the cases where we observed the most significant differences. In general, for the small-size problems, the mean size of HN is only slightly smaller than the mean size of N, but for the medium-size and the large-size problems it is significantly smaller. The max size of the neighborhood is up to 48 times smaller for the small problems, up to 278 times for medium, and up to 1331 for the large ones. Moreover, we experimentally observed that very often Walkplan with HN computes a solution performing a number of search steps lower than or similar to those performed by using the basic search neighborhood.

In Section 6.2 we will show that our local search procedure performs significantly better with the heuristic neighborhood than with the basic neighborhood.

4.5. Heuristics for generating good quality solution NA-graphs

Each action in a numerical planning problem is associated with a cost, which is defined by the problem objective function and the numerical information in the current graph. More precisely, given a planning problem Π and an NA-graph $\mathcal{A}$ for Π, the cost of an action a at a level l of $\mathcal{A}$ (and of the corresponding action node) is defined as follows

$$Cost(a, l) = (f \circ \xi_1 \circ \cdots \circ \xi_k)(\bar{x}_l) - f(\bar{x}_l),$$

where f is the plan metric function of Π (see Definition 4 in Section 2), $\xi_j \in NE(a)$, $j = 1 \ldots k$, and $k = |NE(a)|$. The cost of an action depends on the graph level l because, in general, it depends on the world state where the action is applied, i.e., the world state identified with level l. Note that the cost of an action can be negative. This is the case when the effects of the action modify the value of some numerical fluent(s) in such a way that the plan metric function improves its value with respect to the problem objective function.

The *quality of the plan* represented by an NA-graph is defined by the value μ of f applied to the world state identified with the goal level of the graph.[9]

When the planning problem description includes an objective function for expressing plan quality, the heuristic function for evaluating the graphs in the search neighborhood includes a term estimating their *execution cost* ($ExecCost(a)$). This term is useful to discriminate the elements in the neighborhood also taking the quality of the plan under construction into account. The execution cost for the NA-graph obtained by adding action node a at level l ($ExecCost(a)^i$) estimates the additional cost (defined by the plan metric) that would be added to the cost of the current plan if a and the other actions in the relaxed plan defining $SearchCost(a)^i$ were added at level l. Similarly for the execution cost of the NA-graph obtained by removing a ($ExecCost(a)^r$). In particular, $ExecCost(a)^i$ is defined as

$$ExecCost(a)^i = \sum_{a' \in \pi_{\mathsf{R}}^i} Cost(a', l)$$

where π_{R}^i is the bag of actions forming the relaxed plan computed by $EvalAdd(a)$ to estimate $SearchCost(a)^i$, as described in Section 4.3, and $Cost(a', l)$ is computed using the values of the numerical fluents in the world state identified with the graph level l where node a is added. $ExecCost(a)^r$ is defined in a similar way: we sum the actions in the relaxed plan π_{R}^r computed by $EvalDel(a)$ and remove the cost of the (deleted) action represented by a. Obviously, the role of the execution cost term in the heuristic function is not improving the accuracy of the search cost term in order to compute a valid plan more quickly, but guiding the search towards a solution with good quality. We empirically observed that often without this term the search produces a first plan with worse quality. However, since Walkplan is a local search method, there is no guarantee that the computed plan is optimal, and in many cases the quality of the

[9] Specifically, it is equal to μ when the objective function is a minimization; while it is equal to $-\mu$ when the objective function is a maximization.

InitializeGraph($\Pi, \mathcal{S}, \sigma_\psi, n$)
Input: A numerical planning problem Π, an NA-graph $\mathcal{S}$ for Π,
a new numerical goal σ_ψ for Π, and a number n of modifications for $\mathcal{S}$.
Output: $\mathcal{S}$ modified by σ_ψ and the addition/removal of n helpful/harmful action nodes.

1. $Pre(a_{end}) \leftarrow Pre(a_{end}) \cup \{\sigma_\psi\}$; /* *the problem goals are extended with* σ_ψ */
2. $A \leftarrow$ set of action nodes added to or removed from $\mathcal{S}$ to generate $N(\mathcal{S})$;
3. $N \leftarrow$ set of NA-graphs in $N(\mathcal{S})$;
4. $m \leftarrow 0$; $M \leftarrow \emptyset$;
5. **while** $m < n$ **do** /* *loop for selecting n action nodes to add/remove to/from* $\mathcal{S}$ */
6. $c_N \leftarrow \sum_{a \in A} Abs(Cost(a))$;
7. $x \leftarrow 0$; $y \leftarrow$ Random$(0, c_N)$; $G \leftarrow N$;
8. **while** $x < y$ **do** /* *loop for selecting an action node to add/remove to/from* $\mathcal{S}$ */
9. $\mathcal{A} \leftarrow$ an NA-graph in G; $G \leftarrow G - \{\mathcal{A}\}$;
10. $a \leftarrow$ action node added to $\mathcal{S}$ or removed from $\mathcal{S}$ to derive $\mathcal{A}$;
11. $x \leftarrow x + Abs(Cost(a))$;
12. **if** $x \geqslant y$ **then** /* *action node a is selected (and cannot be selected again)* */
13. $M \leftarrow M \cup \{a\}$; $A \leftarrow A - \{a\}$; $N \leftarrow N - \mathcal{A}$;
14. $m \leftarrow m + 1$;
15. Modify $\mathcal{S}$ by adding (removing) the selected helpful (harmful) action nodes in M.

Fig. 6. An algorithm for computing the initial NA-graph during the incremental search of good quality plans. Random$(0, c_N)$ returns a random value uniformly distributed between 0 and c_N; *Abs* is the standard absolute value function. The use of *Abs* allows us to informally treat actions with negative costs and with positive costs in the same selection mechanism.

first plan generated can significantly be improved. In order to find high quality plans, we use an incremental method that generates a sequence of valid plans with increasing quality by re-running Walkplan multiple times. Each time we run a new search, we use an algorithm for setting the initial search state (NA-graph) that makes some particular graph modifications to the *last* solution NA-graph $\mathcal{S}$ that has been computed. There are two types of modification to $\mathcal{S}$:

- the preconditions of a_{end} are extended with a new precondition forcing that the quality of the next solution graph (if found) is better than the quality of $\mathcal{S}$;
- some action nodes are added to or removed from $\mathcal{S}$, preferring those that have a bigger impact of the plan quality of $\mathcal{S}$ (i.e., we prefer to remove actions that are highly expensive, and to add actions that significantly reduce the overall plan execution cost).

The selection of the action nodes for modifying $\mathcal{S}$ is done by a method that is very similar to the roulette wheel selection technique used in genetic algorithms for forming pairs of chromosomes [23]. Fig. 6 gives our algorithm for computing the initial NA-graph during the incremental plan quality optimization (InitializeGraph). The algorithm first adds a new precondition node σ_ψ to a_{end} (step 1), which represents the following numerical precondition:

$$\psi(\overline{X}) = \begin{cases} \top & \text{if } f(\overline{x}^*_G) - f(\overline{X}) > 0 \\ \bot & \text{otherwise} \end{cases}$$

where X is the set of the problem numerical fluents, f is the plan metric function, and $\overline{x}^*_G$ is the numerical information in the world state identified with the goal level of $\mathcal{S}$. It is easy to see that the modified NA-graph $\mathcal{S}$ is not a solution graph anymore. In particular, goal σ_ψ is not supported, and it imposes the new constraint that the solution NA-graph reached from $\mathcal{S}$ (if any) has an execution cost lower than the execution cost of the original $\mathcal{S}$. Since plan metric function f has to be minimized, the search neighborhood of the modified $\mathcal{S}$ for the (flawed) goal level l_G consists of the NA-graphs obtained by adding the helpful action nodes for l_G with negative costs, and removing the harmful action nodes for l_G with positive costs. InitializeGraph identifies such action nodes, evaluates their costs, and selects n of them.[10] Finally, the selected helpful action nodes are added to the modified $\mathcal{S}$ and the selected harmful action nodes are removed from it. Each of these action nodes is independently selected with a probability that is proportional

[10] The value of parameter n is automatically set to a default value, which depends on the size of the search neighborhood, and was empirically determined. For more details the interested reader can see [18].

to the absolute value of its effect on the execution cost of the plan (with the constraint that each action is selected at most once). As we prove in [18], this favors choosing graph modifications that have the biggest impact on the quality of the plan.

Concerning the complexity of InitializeGraph, we observe that it is polynomial in n and the maximum size of $N(\mathcal{S})$. In practice, in all our experimental tests, the computational cost of the graph initialization was negligible with respect to the overall cost of the planning process. In Section 6.2, we will show that InitializeGraph helps the optimization process to find good quality plans more quickly.

5. Handling time in domains with numerical fluents

In this section, we briefly describe how the plan representation and search methods for temporal planning given in [15] are integrated with the representation and search techniques for numerical planning presented in this paper.

5.1. Extended plan representation

For PDDL2.1 domains with "durative actions" [11], the NA-graph is augmented with some temporal information required for representing and computing a temporal schedule of the plan actions. When actions have durations, the Ω-set of temporal constraints in an NA-graph $\mathcal{A}$ consists of ordering constraints between action *endpoints* and additional constraints specifying the (static or dynamic) durations of the actions in the plan π represented by $\mathcal{A}$. As described in [14], the particular action endpoints in an ordering constraint depend on the type of precondition ("`at start`", "`over all`" or "`at end`") or effect ("`at start`" or "`at end`") involved in the causal relation or mutex relation generating the constraint. The special actions a_{start} and a_{end} are associated with the special variable s, which is constrained to precede every action endpoint, and the special variable e, which is constrained to follow every action endpoint, respectively. The constraints in Ω form a temporal CSP, which is a particular STP [6] that, by construction, is consistent (i.e., it admits at least one solution). This STP is dynamic (at each search step new constraints and variables can be added or removed), and is solved at each search step. The solution of the STP that assigns 0 to s and the earliest possible time value to e is used to label the action nodes in $\mathcal{A}$ and to schedule the start times of the actions in π. The time value assigned to e defines the makespan of the plan, which is represented by the special numerical fluent `total-time`.

Moreover, each propositional fact node (numerical precondition node) in $\mathcal{A}$ is labeled by the earliest time when the corresponding propositional fact (numerical precondition) becomes true; while each numerical fluent node at a level l of $\mathcal{A}$ is labeled by the *latest* time when the action at a level $i < l$ affects the corresponding numerical variable.

In a temporal domain with numerical fluents, the duration of an action can depend on the world state where the action is executed (e.g., the duration of the "refuel" action depends on the current level of the fuel in the vehicle tank). In order to deal with these dynamic durations, at each search step, before solving the temporal CSP of $\mathcal{A}$, for each level l of the graph we update the duration of the action at that level (and the corresponding duration constraint) with respect to the world state identified with l.

5.2. Search neighborhood and heuristics

For numerical planning problems with action durations, the definitions of the local search neighborhood and heuristic search neighborhood given in Sections 4.2 and 4.4 remain the same, except for the special temporal precondition, defined in the next subsection, imposing a constraint on the plan makespan. For this particular precondition, which is added to a_{end} when the search is re-initialized, the neighborhood is obtained by removing an action node that is on a *critical path of* a_{end}, i.e., on a path from a_{end} that determines the makespan of the plan represented by the current NA-graph.

When the plan metric function involves `total-time`, the elements of the search neighborhood are evaluated according to a linear combination of three terms, instead of two. In addition to the search and execution costs of the neighborhood element, we also evaluate its *temporal cost*. As in [15], the temporal cost is an estimate of the contribution that the addition of the actions in the relaxed plan π_{R} evaluating the search cost would give to the makespan of the plan under construction. The temporal cost is computed by estimating the end time of π_{R}. This can be done by extending RelaxedNumPlan with some steps estimating the earliest time when the preconditions

and effects of the relaxed plan actions become true. The only significant difference concerns computing the time estimate for the *numerical* (sub)goals of the relaxed plan actions, which in [15] are not considered. When a numerical (sub)goals is satisfied in the initial world state of the relaxed plan, and at the corresponding graph level l there is no node representing it,[11] we heuristically evaluate its temporal value. We do this by using the maximum value over the temporal values associated with the numerical fluents involved in the numerical condition. Such a time corresponds to the end time of the *last* action at a level preceding l and affecting one of the fluents in the condition. For example, let 20 and 0 be the numerical values labeling fluent nodes x and y at level l, and 10 and 0 the temporal values labeling them, respectively. Then, the numerical condition $x + y - 20 \geqslant 0$ holds in the state identified with l and its temporal value is estimated to be 10.

5.3. Search re-initialization

We distinguish the cases in which (i) the objective function only requires minimization of `total-time`, and (ii) it requires minimization of `total-time` linearly combined with other numerical fluents. In case (i), in order to compute a plan with better quality, we extend the preconditions of a_{end} with a special numerical precondition imposing that (T − `total-time`) is greater than 0, where T is the makespan of the last valid plan computed, and we re-initialize the search using an algorithm similar to InitializeGraph (Section 4.5). In this case, the roulette wheel selection method of InitializeGraph prefers the graph modifications in the neighborhood that remove the actions with the greatest durations on a critical path for a_{end}.

In case (ii), as for purely numerical problems, we extend the preconditions of a_{end} with a special numerical precondition forcing that the quality of the next solution NA-graph is better than the quality of last solution graph $\mathcal{S}$. However, in this case we re-initialize the search using the roulette wheel selection method twice. First we consider the modifications affecting only the makespan; then we consider the modifications affecting only the non-temporal terms of the plan metric function. The motivation of this is making the definition of the costs associated with the graph modifications defining the neighborhood for the new special precondition easier to evaluate. Specifically, let n be the total number of graph modifications for reinitializing $\mathcal{S}$ (n is computed as described in Section 4.5). First we select n' graph modifications among those removing an action on a critical path for a_{end}, where the value of n' depends on the contribution that the value T of `total-time` in $\mathcal{S}$ gives to the plan metric function.[12] Then we select the remaining $n - n'$ graph modifications among those forming the search neighborhood for the special additional numerical goal described in Section 4.5 (which does not involve `total-time`).

6. Experimental analysis

In this section, we present an experimental study aimed at testing the effectiveness of the proposed techniques. In the first short subsection, we describe the experimental settings and the overall results. In the second subsection, we experimentally investigate the impact of the main heuristic techniques that we have proposed on the overall performance of our approach. Finally, in the third subsection, we compare our planner and three other state-of-the-art satisficing planners in terms of number of solved problems, only CPU-time, only plan quality, and both CPU-time and plan quality. Here we present and discuss the general results of the experimental comparison (in [18] we give plots of all the performance data, while Appendix A contains a selection of them). In order to show that the observed performance gaps are statistically significant, we also use the Wilcoxon test [47] as a summarization of these experimental results. Moreover, we test the performance of our approach relative to the other considered planners for a class of problems with tight resources in a practically useful known domain.

6.1. Experimental settings and overall results

We implemented our techniques for numerical planning in a new version of a planner called LPG-td, incorporating various additional techniques for managing other features of PDDL2.2. In the rest of the paper, "Metric-LPG" indicates

[11] This can happen because the preconditions of the actions in the relaxed plan may not be preconditions of actions in the current NA-graph.

[12] n' is $\lceil n \cdot \frac{\alpha T}{\alpha \cdot T + |g(\bar{x}^*_G)|} \rceil - 1$, if $T > 0$, and 0 otherwise; α is the coefficient of `total-time` in the plan metric function f, $g = f - \alpha T$, and $\bar{x}^*_G$ is the numerical information in the world state identified with the goal level of $\mathcal{S}$.

the set of the implemented techniques that are relevant for planning with numerical fluents and multi-criteria objective functions.[13]

Since Metric-LPG is an incremental planner, in the comparison with the other tested planners, we evaluate its performance with respect to two different main criteria: CPU-time required to compute a valid plan (indicated with Metric-LPG.s), and quality of the best plan generated within a given CPU-time limit (indicated with Metric-LPG.q). Often the best quality solution computed by Metric-LPG requires much more time than the first (unoptimized) solution. However, it should be noted that when a planning problem can be solved by a satisficing planner within a reasonable amount of CPU-time (e.g., 10 CPU-minutes), the quality of the computed solutions is practically very important, and for many applications it can be more meaningful than CPU-time. Moreover, Metric-LPG generates a sequence of valid plans with increasing quality, and often it produces good quality *intermediate* plans much more quickly than the best (last) plan. In Section 6.2, we give empirical evidence about this behavior. In general, in Metric-LPG there is a tradeoff between plan quality and CPU-time that can be exploited by the user, depending on whether for the application domain under consideration planning speed is more important than plan quality, or vice-versa.

The other tested planners that support PDDL numerical fluents and are publicly available are: Metric-FF [27], MIPS (version 3) [9], and SGPLAN5 (version 5.2.2) [4,32]. We ran all planners with the same default settings for every problem attempted. We conducted all the experimental tests using an Intel Xeon(tm) 3 GHz machine, with 1 Gbytes of RAM. Unless otherwise specified, the CPU-time limit for each run was 10 minutes, after which termination was forced. Since our planner uses a randomized search algorithm, the results that we give for our planner are median values over five runs for each problem considered. When Metric-LPG exceeded the CPU-time limit for more than two of the five runs, we consider the problem unsolved.

As test domains we used the metric and metric-temporal variants of the domains developed for the 3rd, 4th and 5th international planning competitions (IPC-3, IPC-4 and IPC-5, respectively) [7,28,36]: `Depots`, `Driverlog`, `Openstacks`, `Pathways`, `Rovers`, `Satellite`, `Settlers`, `TPP`, `Umts` and `ZenoTravel`.[14] The number of the benchmark problems in these domains is 644.

Overall, the results indicate that the techniques proposed in this paper are effective and have a significant positive impact on the performance of our approach. Our planner is generally quite competitive with other existing powerful planners handling numerical planning problems, and it performs very well especially in terms of plan quality. According to the Wilcoxon test, overall, for the large set of benchmark problems that we considered, Metric-LPG outperforms every other tested planner in terms of either CPU-time or plan quality, and in some cases in terms of both (in particular, Metric-LPG.s is generally faster than SGPLAN5 and computes better quality solutions). Moreover, the results concerning numerical problems with tight resources indicate that our approach is more suitable than other satisficing planners for solving this class of problems, which confirms a similar recent experimental observation by Hoffmann and others for a different set of benchmark problems [30].

However, we would like to remark that, while the goal of our analysis is giving substantial empirical evidence of the effectiveness of our techniques for numerical planning, the experiments are not aimed at showing that a planner is "absolutely" better than another one.

6.2. An experimental study of some heuristic techniques in metric-LPG

In this section, we experimentally investigate the impact of some heuristic techniques that we have presented on the performance of Metric-LPG. In order to have some insights about their effectiveness, we compare them with other possible alternative techniques. In particular, we evaluate: (1) the new heuristic evaluation function E versus a simpler heuristic function ignoring numerical information; (2) the heuristic search neighborhood HN versus the basic one (N); (3) the combined use of E and HN versus the combination of N and a very simple (but very fast to compute) evaluation function; and (4) the incremental plan optimization process using algorithm InitializeGraph

[13] LPG-td is written in C and is available from http://lpg.ing.unibs.it.

[14] The objective function of the numerical problems considered in our experiments involves one or more numerical fluents, and, for the metric-temporal variants, also the special fluent `total-time`. The benchmark problems in the metric-temporal variants are numerical planning problems extended with some PDDL2.1 temporal features. We have not considered the IPC-4 `Promela` domain because this domain contains numerical preconditions involving equality conditions, which currently our planner does not explicitly support. The IPC-3 benchmark problems that we considered are all those developed for the fully-automated planners and the hand-tailored planners.

versus the empty plan initialization strategy. Moreover, in the related work section, we will give some experimental evidence that the performance of our planner is significantly improved with respect to a previous version including some preliminary techniques for numerical planning that are abstractly described in [15].

The alternative functions for choosing an element in the neighborhood that we consider are:

- E_0, which randomly selects an element in the neighborhood;
- E_1, which is similar to E with the difference that numerical preconditions are ignored.

6.2.1. Heuristic evaluation function

Fig. 7 and Table 2 give an overall picture of the compared techniques in terms of number of solved problems within a CPU-time limit ranging from 0.01 seconds to 600 seconds (considering the first solution found). Concerning E versus E_1, under the same neighborhood definition (*HN*), for small CPU-time limits (lower than 30 milliseconds) Metric-LPG performs similarly with either E or E_1. However, for higher CPU-time limits, the use of E allows the planner to solve many more problems. In particular, with 600 CPU-seconds, the planner solves 114 more problems.

Table 2 shows the distribution of the solved problems over the different benchmark domains. For `Pathways`, `Settler` and `TPP` the use of E allows the planner to solve many more problems than when using E_1. For any other domain, the number of problems solved using E is slightly greater, with the exception of `DriverLog`, `Openstacks` and `Umts`, where the planner solves all problems with any of the compared functions. This is not surprising because, in `DriverLog` there is no numerical precondition or goal (and hence evaluating the neighborhood with either E or E_1 makes no difference), while in `Openstacks` and `Umts`, computing a valid plan is a very easy task for our planner (the main challenge for the problems in these domains is computing high quality plans).

The plot of Fig. 8 gives a graphical representation of the performance of E compared to E_1 in terms of the overall CPU-time for solving numerical planning problems. Each cross corresponds to a problem solved using either E or E_1. If a cross is above the solid diagonal, then E performs better than E_1, otherwise E_1 performs better than E. The distance of a cross from the main diagonal indicates the performance gap (the greater the distance, the greater the gap). It easy to see that, in general, E allows the planner to find a plan more quickly, and often the performance gap is at least one order of magnitude.

Observation 1. *For the set of the benchmark problems considered,* Metric-LPG *with E performs significantly better than with E_1 in terms of both number of solved problems and plan generation speed.*

The previous observation suggests that numerical information should be considered in the definition of our heuristic evaluation function, even if this makes the function more complex and computationally more expensive.

6.2.2. Heuristic search neighborhood

We evaluate the effectiveness of the heuristic neighborhood with respect to the basic one when combined with the new evaluation function E. The results of the analysis in Fig. 7 show that, for every CPU-time limit considered, the planner solves more problems using *HN* than using *N*. In particular, when the CPU-time limit is 600 seconds, the use of *HN* allows the planner to solve 123 more problems. Table 2 shows the distribution of the solved problems over the different benchmark domains. For `Pathways`, `Settler`, `TPP` and `ZenoTravel`, Metric-LPG with *HN* solves many more problems than with *N*. For each other tested domain, the number of problems solved using *HN* is slightly greater than using *N* (with the exception of `Openstacks` and `Umts`, where the planner performs similarly with either of the two neighborhood definitions).

Moreover, the comparison of *HN* and *N* is in favor of *HN* also in terms of plan generation CPU-time. Since most of the circles in the plot of Fig. 8 are above the solid diagonal, we can conclude that, for our benchmark problems, Metric-LPG with *HN* is generally faster.

Observation 2. *For the benchmark problems considered,* Metric-LPG *with neighborhood HN evaluated by function E performs significantly better than with neighborhood N in terms of both number of problems solved and plan generation speed.*

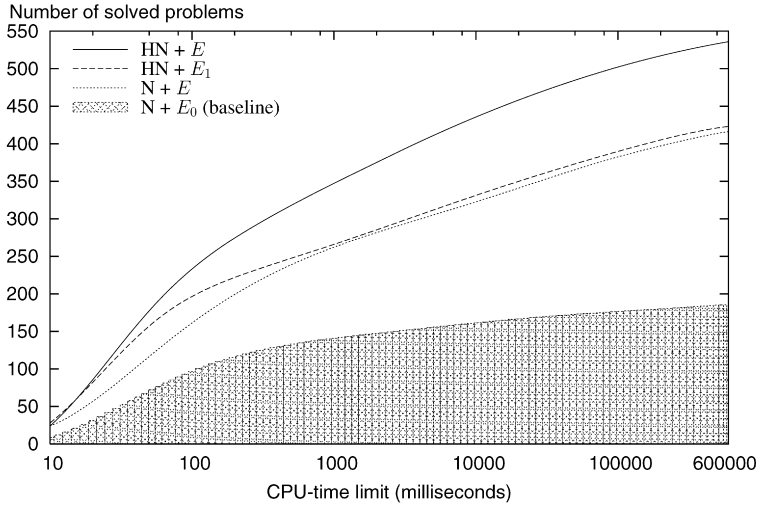


Fig. 7. Number of problems solved by Metric-LPG over the 644 test problems with respect to an increasing CPU-time limit using different combinations of the neighborhood definition (*N* or *HN*) and heuristic evaluation function (E, E_0 or E_1). The gray area corresponds to the simplest (but fastest to compute) strategy using *N* and E_0. The CPU-time limit on the x-axis is in logarithmic scale.

Table 2
Distribution of the solved problems over the benchmark domains for the different combinations of neighborhood definition and evaluation functions considered

Domains	#Problems	$HN + E$	$HN + E_1$	$N + E$	$N + E_0$
Depots	42	24	21	20	1
DriverLog	80	40	40	37	2
Openstacks	20	20	20	20	10
Pathways	30	30	10	8	4
Rovers	120	116	103	111	30
Satellite	72	54	44	48	21
Settlers	20	16	1	0	0
TPP	80	74	35	28	10
Umts	100	100	100	100	100
ZenoTravel	80	65	61	44	8
Total	644	539	425	416	186

This empirical observation has an intuitive explanation. Evaluating the basic neighborhood using an accurate function like E can make each search step computationally too expensive: the heuristic power of the function does not pay off when the portion of the search space explored by the search is too limited. On the other hand, the use of E combined with a reduced neighborhood, where bad elements have been filtered out, is significantly effective for the search.

6.2.3. Heuristic evaluation function combined with the heuristic neighborhood

We evaluate the combination of *HN* and E versus the simple use of N and E_0. As expected, the results in Fig. 7 and Table 2 show that $HN + E$ performs much better. On the other hand, in some test domains the performance of $N + E_0$ seems surprising good, given that in E_0 there is no heuristic information. The main reason of this behavior is that the IPC benchmark problems contains many problems that are quite easy to solve by generating a sub-optimal plan (at least with our approach), while they are much harder to solve by computing high quality plans. In particular, all problem in the Umts domain have this nature. The local search procedure with E_0 does not incur in local minima, and for small or trivial problems, it is sufficient to perform many search steps to "accidentally" find a solution NA-graph.[15]

[15] Interestingly, this behavior seems peculiar of our search space. We experimentally observed that the use of E_0 in a state-space forward local search is less effective than in our NA-graph based local search.

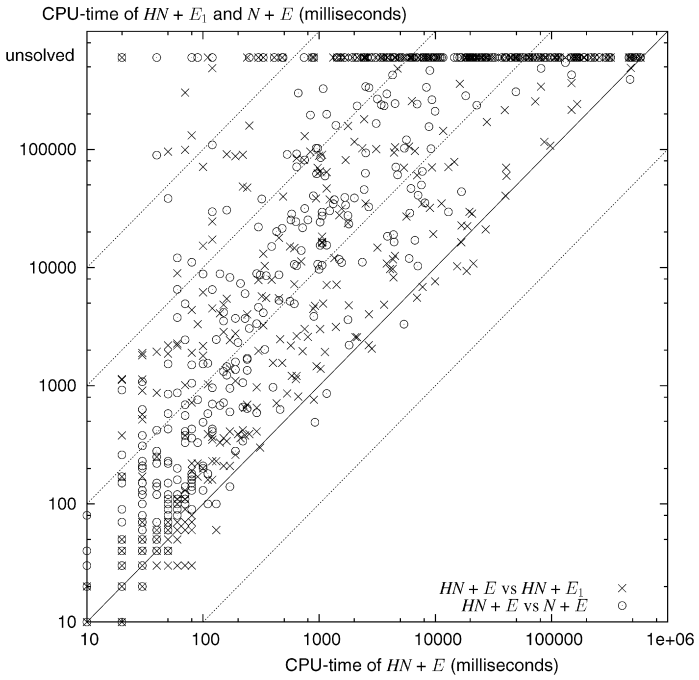


Fig. 8. Scatterplot comparing the performance of heuristic functions E versus E_1, and neighborhoods *HN* versus *N*. On the x-axis, we have the CPU-milliseconds (log scale) of Metric-LPG with $HN + E$; on the y-axis we have the CPU-milliseconds (log scale) of the planner with $HN + E_1$ (crosses) and $N + E$ (circles).

Observation 3. *For the benchmark problems considered,* Metric-LPG *using* E_0 *and N only solves small or trivial problems.*

6.2.4. Tradeoff between CPU-time and plan quality in Metric-LPG

Metric-LPG is an incremental planner producing a sequence of valid plans with increasing quality. This is a useful feature of our planner that allows the user to tradeoff plan quality and CPU-time, depending on the needs of her/his application: the more CPU-time is given to the planning process, the better the plan quality is (up to the bound defined

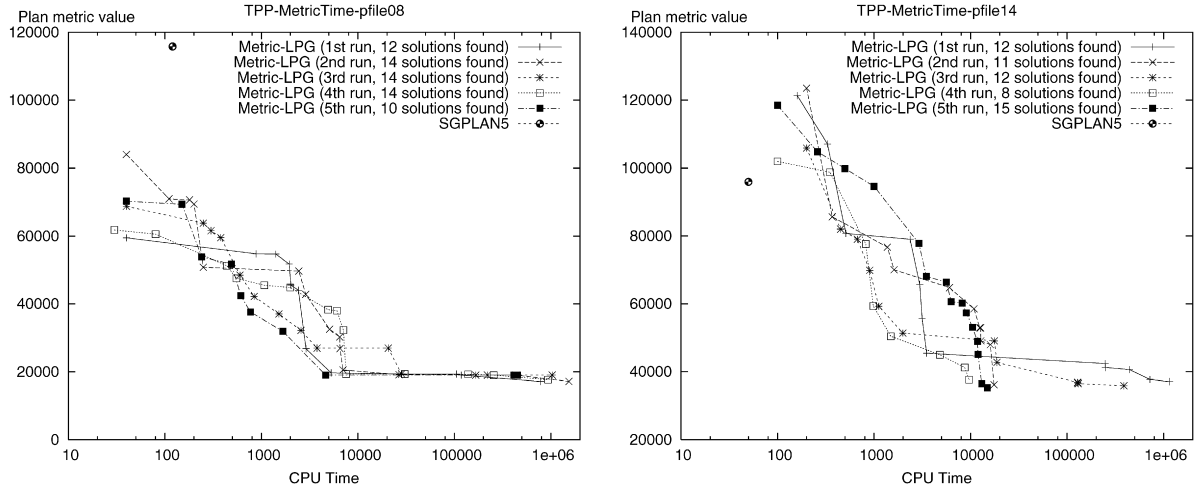


Fig. 9. Plan quality and corresponding CPU-milliseconds (logarithmic scale) for the plans generated by SGPLAN5 and by five runs of Metric-LPG for two problems in the metric-temporal variant of TPP. The plan metric function to minimize is a linear combination of the total cost of the purchased goods and the plan makespan.

by the optimal plan quality). As an example of this behavior, the two plots of Fig. 9 show the sequences of solutions computed by five runs of Metric-LPG solving two problems in the metric-temporal variant of domain TPP [7] (p08 and p14). These plans are compared with the corresponding plans computed by SGPLAN5. Concerning the plot on the left hand side of Fig. 9, the first solution computed by Metric-LPG is always better than the solution generated by SGPLAN5 (and is computed using less CPU-time). Moreover, in most cases the initial plan is considerably improved by the next plans that are generated by Metric-LPG within about one additional CPU-second. Concerning the plot on the right hand side of Fig. 9, SGPLAN5 very quickly computes a plan that is better than the first plan generated by our planner (which is also slower). However, Metric-LPG computes several additional plans that are better than the plan computed by SGPLAN5, and require much less CPU-time than the last (best) solution computed by Metric-LPG.

In order to better understand the tradeoff between plan quality and CPU-time for the intermediate solutions generated by Metric-LPG, we conducted the following experimental analysis. We considered the best plan computed within 600 CPU-seconds (Metric-LPG.q) and we compared the quality of some intermediate solutions with this plan. The analysis involves all the 644 benchmark problems. Fig. 10 gives a graphical representation of the results for intermediate solutions computed 10, 60, and 240 seconds after the time when the first plan is generated. In each plot, the

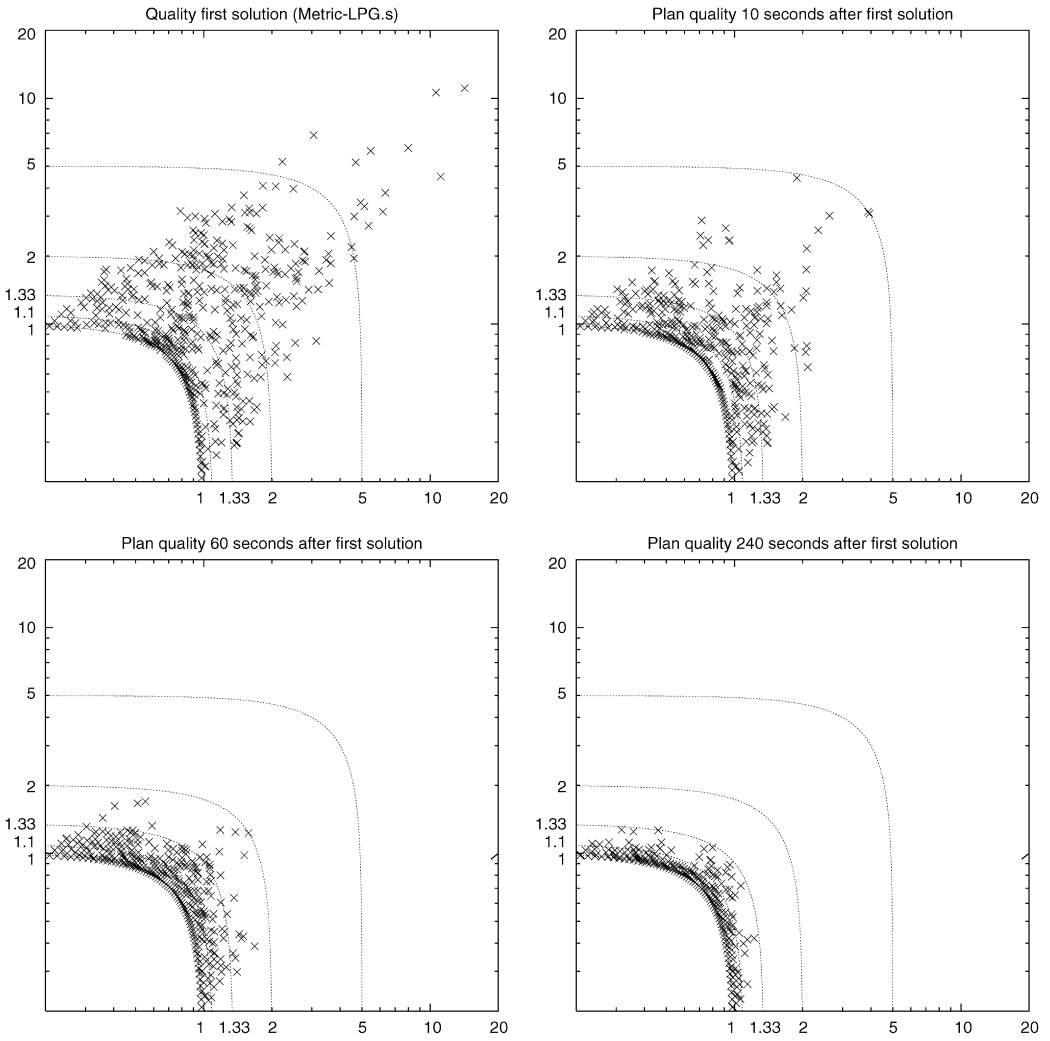


Fig. 10. Quality of some intermediate plans computed by Metric-LPG with respect to the best plan generated within 600 CPU-seconds (curve with radius 1): first plan generated by Metric-LPG.s, best plan 10 seconds after the first plan, best plan 60 seconds after the first plan, and best plan 240 seconds after the first plan. Each cross corresponds to a plan. The plots are in polar coordinates: the distance of a cross from the curve with radius 1 shows the plan quality ratio, the angle is (arbitrarily) determined in a way that spreads the crosses to make the plot data more readable.

curve with radius 1 corresponds to the quality of the plan computed by Metric-LPG.q, while each cross corresponds to the quality of an intermediate plan. The (minimum) distance of a cross from the curve with radius 1 is the ratio between the quality of the plan corresponding to the cross and the quality of the best plan (Metric-LPG.q) solving the same problem.

As demonstrated by the results in the top-left plot of Fig. 10, for our benchmark problems often the first plan computed by Metric-LPG (Metric-LPG.s) has a quality that is similar to, or at most 33% worse than the plan computed by Metric-LPG.q. (This happens especially for the smallest problems, where the first solution has already a high quality, and for some of the largest problems, where the incremental process of Metric-LPG does not improve the first plan.) However, many plans are significantly worse than the best reference plan.

If we give 10 seconds to the incremental plan optimization process (top-right of Fig. 10), the quality of the generated plans significantly improves. There is no plan that is more than 5 times worse than the reference plan, and only few plans are between 2 and 5 times worse. If we give 60 seconds to the incremental plan optimization process (bottom-left of Fig. 10), the quality of the generated plans significantly improves again. Now most of the plans are similar to the reference plan, and only few of them have a quality that is 1/3 worse than the quality of the reference plan. Finally, if we give 240 seconds to the incremental plan optimization process (bottom-right of Fig. 10), the quality of the large majority of the plans is the same as, or very similar to the quality of the reference plan (most of the plans are no worse than 1/10 of the quality of the reference plan).

Observation 4. *The incremental plan optimization process of* Metric-LPG *can significantly and quickly improve the quality of the first generated plan.*

Interestingly, our plan optimization process could also be used to improve the plans generated by *other* planning systems.

6.2.5. Search re-initialization strategy

We evaluate the proposed plan re-initialization strategy using algorithm InitializeGraph for the incremental plan optimization process of our planner. We call this strategy PlanRestart. In order to better understand the importance of PlanRestart for the incremental optimization of Metric-LPG.s, we also ran our planner using the alternative simple plan re-initialization strategy that always uses the empty plan, which we call EmptyRestart.

The results of this analysis are in Table 3 and Fig. 11. Table 3 shows the relative performance of Metric-LPG using PlanRestart and EmptyRestart with a CPU-time limit of 10 CPU-seconds for the incremental optimization process. Most of the plans computed using PlanRestart are better than those computed using EmptyRestart, and in DriverLog, Rovers, Satellite and Openstacks, they are much better.

Table 3
Performance of PlanRestart versus EmptyRestart with 10 CPU-seconds of incremental plan optimization. Number of problems solved by Metric-LPG (2nd column); number of initial plans improved by either of the two strategies (3rd column); and relative plan improvements of the two strategies: number of problems for which using PlanRestart gives plans that are at least 33% worse (4th column), worse/better (5th/6th columns), at least 33% better (7th column) and at least 2 times better (8th column) than the plans obtained using EmptyRestart

Domain	Problem solved	Improved plans	33%W	W	B	33%B	100%B
Depots	24	18	0	5	6	1	0
DriverLog	40	36	0	3	21	6	1
Openstacks	20	18	0	4	6	1	0
Pathways	30	9	0	0	1	0	0
Rovers	116	92	0	14	38	7	1
Satellite	54	47	0	2	41	11	1
Settlers	16	6	0	1	1	0	0
TPP	74	42	0	17	16	2	0
Umts	100	60	0	3	1	0	0
ZenoTravel	65	43	0	7	26	7	0
Total	539	371	0	56	157	35	3

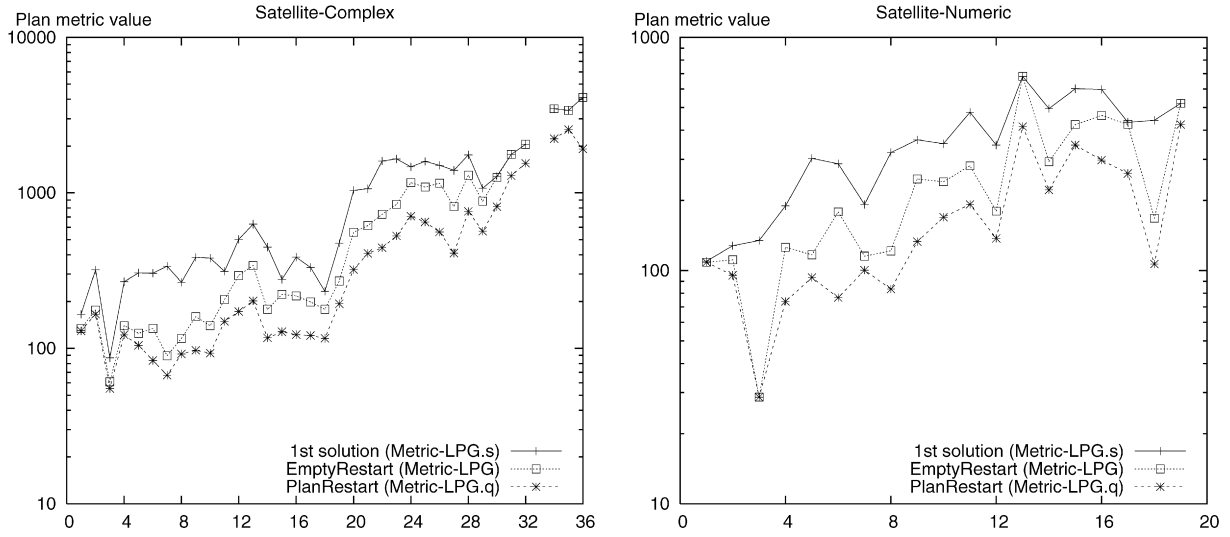


Fig. 11. Plan quality of the first solution found by Metric-LPG (Metric-LPG.s) and of the best solution found by Metric-LPG (Metric-LPG.q) using either PlanRestart or EmptyRestart within 600 CPU seconds for two variants of domain `Satellite`. On the x-axis we have the problem names simplified by numbers; on the y-axis we have the plan metric function (the lower the better) in logarithmic scale.

Observation 5. *For a large subset of the benchmark problems considered,* PlanRestart *is more effective* (*helps the optimization process finding good quality plans more quickly*) *than* EmptyRestart.

Often PlanRestart allows the optimization process to find better quality plans more quickly than EmptyRestart, but given the stochastic nature of our local search procedure, it does not always perform better than EmptyRestart. Moreover, in very long runs it is possible that the two compared strategies compute similar solutions. However, even with a CPU-time limit of 600 seconds, for some domains we still have a significant performance gap. For instance, Fig. 11 shows the plan quality of the first generated plan and the best generated plan using either PlanRestart or EmptyRestart for two variants of a benchmark domain (`Satellite`). All plans computed using PlanRestart are better than or no worse than those computed using EmptyRestart, and many of them are much better.

6.3. Metric-LPG versus the other satisficing planners

In this section we compare the performance of Metric-LPG with respect to three state-of-the-art planners (Metric-FF, MIPS and SGPLAN5) in terms of number of solved problems, only CPU-time, only plan quality, both CPU-time and plan quality, and capability of solving a class of problems with tight resources.

6.3.1. Number of solved problems

Fig. 12 shows the results of an analysis comparing different planners in terms of number of solved problems within a CPU-time limit ranging from 10 milliseconds to 600 seconds. Overall, Metric-LPG solves more problems than the other compared planners. When the CPU-time limit is very low (below 30 milliseconds), SGPLAN5 solves more problems than Metric-LPG. However, for greater CPU-time limits Metric-LPG solves many more problems than any other compared planner. In particular, when the limit is 600 CPU-seconds, Metric-LPG solves 539 problems, Metric-FF 157, MIPS 149 and SGPLAN5 477.[16]

Table 4 shows the distribution of the solved problems over the benchmark domains with respect to the number of problems that can be attempted by the planner, which depends on the PDDL features involved in the problems and supported by the planner.

[16] Metric-FF solves a significant lower number of problems than Metric-LPG and SGPLAN5 because it does not support the temporal features that are required in nine of the seventeen domains forming our benchmarks.

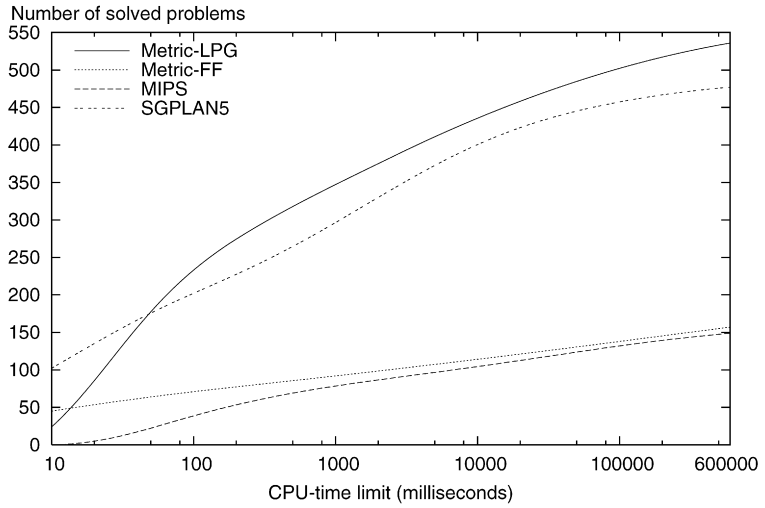


Fig. 12. Number of problems solved by Metric-LPG, Metric-FF, MIPS and SGPLAN5 with respect to an increasing CPU-time limit (from 0.01 to 600 seconds) for our 644 benchmark problems. On the x-axis, we have the CPU-time limit in milliseconds (logarithmic scale); on the y-axis, we have the number of solved problems.

Table 4
Number of problems solved/supported by Metric-LPG, Metric-FF, MIPS and SGPLAN5 for each benchmark domain considered. The CPU-time limit is 600 seconds. "–" indicates no solved problem, because it contains a PDDL feature that is unsupported by the planner; grayed numbers indicate the best performance for each domain

Domain	Metric-LPG	Metric-FF	MIPS	SGPLAN5
Depots	24/42	20/42	7/42	21/42
DriverLog	40/80	34/80	31/80	33/80
Openstacks	20/20	–	–	20/20
Pathways	30/30	–	–	30/30
Rovers	116/120	18/40	20/120	69/120
Satellite	54/72	14/36	27/72	49/72
Settlers	16/20	6/20	–	17/20
TPP	74/80	31/40	–	68/80
Umts	100/100	–	–	100/100
ZenoTravel	65/80	34/40	64/80	70/80
Total	539/644	157/298	149/394	477/624

Observation 6. *For the benchmark problems considered, in terms of number of solved problems,* Metric-LPG *performs generally better than the other tested planners.*

6.3.2. Performance in terms of (only) CPU-time

Fig. 13 shows a graphical representation of the overall performance of Metric-LPG.s with respect to Metric-FF (circles), MIPS (triangles) and SGPLAN5 (crosses) in terms of CPU-time for our benchmark problems. For this analysis, we considered all the problems that can be *attempted* by both the compared planners *and* that are solved by at least one of them.

A circle above (below) the solid diagonal corresponds to a problem where Metric-LPG.s is faster (slower) than Metric-FF. The distance of a circle from the solid diagonal indicates the performance gap (the greater the distance, the greater the gap). Analogously for the triangles and crosses.

The data inside the areas marked "A1" and "A2" in the left bottom corner of the plot correspond to the easiest test problems that are solved by both the compared planners with at most 100 CPU-milliseconds. The plot has additional parallel lines dividing the picture into sectors. A circle over the line labeled "1oF" (under the line labeled "1oS") corresponds to a problem where Metric-LPG.s is at least one order of magnitude faster (slower) than Metric-FF. Similarly,

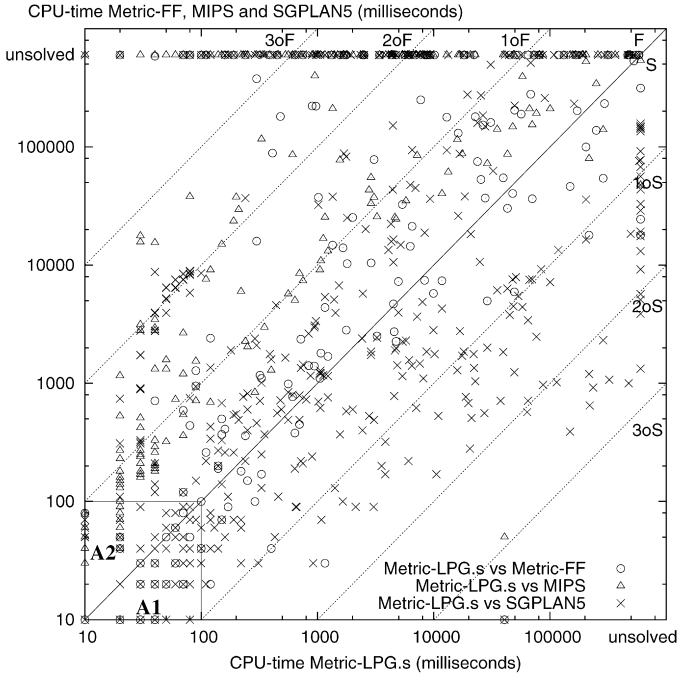


Fig. 13. Scatterplot comparing the performance of Metric-LPG.s versus Metric-FF, MIPS and SGPLAN5 in terms of CPU-time for our benchmark problems. On the x-axis, we have the CPU-milliseconds of Metric-LPG.s (log scale); on the y-axis we have the CPU-milliseconds of the compared planners (log scale). "F/S" means faster/slower. "XoF/S" means at least X orders of magnitude faster/slower.

Table 5
Comparison of Metric-LPG.s versus Metric-FF, MIPS and SGPLAN5 in terms of number of problems for which it is faster/slower (column "F"/"S"), at least one order of magnitude faster/slower (column "1oF"/"1oS"), at least two orders faster/slower (column "2oF"/"2oS"), at least three orders faster/slower (column "3oF"/"3oS")

CPU-time analysis	3oS	2oS	1oS	S (in A1)	F (in A2)	1oF	2oF	3oF
Metric-LPG.s vs Metric-FF	1	0	5	77 (45)	114 (4)	54	30	13
Metric-LPG.s vs MIPS	0	1	2	5 (0)	296 (44)	178	118	62
Metric-LPG.s vs SGPLAN5	1	14	61	286 (141)	250 (18)	131	74	22

the lines labeled "2oF" ("2oS") and "3oF" ("3oS") identify sectors corresponding to problems where Metric-LPG.s is at least two and three, respectively, orders of magnitude faster (slower). Analogously for the triangles and crosses.

The data with the "unsolved" value on the y-coordinate correspond to the problems solved by Metric-LPG and unsolved by the compared planner; the data with the "unsolved" value on the x-coordinate correspond to the problems solved by the compared planner and unsolved by Metric-LPG. Consistently with the analysis of Fig. 12, the data on the "unsolved" y-coordinate are much more dense than the data on the "unsolved" x-coordinate.

Since almost all triangles are over the solid diagonal, the plot gives a visual indication that Metric-LPG.s is generally faster than MIPS. Concerning Metric-LPG.s and Metric-FF, the number of circles above the diagonal is higher than those below the diagonal, and this indicates that, for most of the test problems, Metric-LPG.s is faster than Metric-FF. On the contrary, for Metric-LPG.s and SGPLAN5, this plot gives a much less clear indication, since there are many crosses above the diagonal but also many below it.

Note that in Fig. 13 there are many crosses, as well as triangles and circles, that appear overlapped and cannot be distinguished. Hence, this plot only gives a general coarse picture of the relative performance of the compared planners. Table 5 gives more details, showing the number of circles, triangles and crosses in each sector of the plot. In particular, concerning the comparison of Metric-LPG.s and SGPLAN5, we have that Metric-LPG.s is faster than SGPLAN5 for a slightly lower number of problems (columns "F" and "S"). Interestingly, about half of the problems where SGPLAN5 is faster belong to the easy region A1, while most of the problems where Metric-LPG.s is faster are

Table 6
Results of the Wilcoxon test for the performance of Metric-LPG.s versus MIPS, Metric-FF and SGPLAN5 in terms of CPU-time for our benchmark problems

Metric-LPG.s *vs* Metric-FF		Metric-LPG.s *vs* MIPS		Metric-LPG.s *vs* SGPLAN5	
− 6.07		− 14.2		− 3.29	
< 0.001		< 0.001		< 0.001	
208		301		561	

outside area A2. Moreover, the number of problems for which Metric-LPG.s is at least one, two and three orders of magnitude faster than SGPLAN5 (columns "1oF", "2oF" and "3oF") is much greater than the number of problems for which it is at least one, two and three, respectively, orders of magnitude slower (columns "1oS", "2oS" and "3oS").

In order to understand the significance of the performance gaps between the compared planners, we carried out a statistical analysis using the Wilcoxon sign-rank test (also known as the "Wilcoxon matched pairs test") [47]. This test has also been used by the organizers of IPC-3 to analyze the relative performance of the competing planners [36]. Table 6 shows the results of the Wilcoxon sign-rank test comparing Metric-LPG.s with Metric-FF, MIPS, and SGPLAN5. We considered all the test problems that can be attempted by both the compared planners and that are solved by at least one of them. When a planner does not solve a problem, the corresponding CPU-time is the IEEE arithmetic representation of positive infinity.[17]

The data for carrying out the Wilcoxon test are derived as follows. For each planning problem we compute the difference between the CPU-times of the two planners being compared, defining the samples of the test for the CPU-time analysis. The absolute values of these differences are then ranked by increasing numbers, starting from the lowest value. (The lowest value is ranked 1, the next lowest value is ranked 2, and so on.) Then we sum the ranks of the positive differences, and we sum the ranks of the negative differences. If the performance of the two compared planners is not significantly different, then the number of the positive differences is approximately equal to the number of the negative differences, and the sum of the ranks in the set of the positive differences is approximately equal to the sum of the ranks in the other set. Intuitively, the test considers a weighted sum of the number of times one planner performs better than the other. The sum is weighted because the test uses the performance gap to assign a rank to each performance difference.

Each cell in Table 6 gives the result of a comparison between the performance of Metric-LPG.s and another tested planner in terms of CPU-time. In this table, as well as in the next ones concerning an analysis based on the Wilcoxon test, when the number of samples is sufficiently large, the T-distribution used by the Wilcoxon test is approximately a normal distribution. Therefore, the cells of the figure contain the z-value and the p-value characterizing the normal distribution. The higher the z-value, the more significant the difference of the performance is. The p-value represents the level of significance in the performance gap. We use a confidence level of 99.9%; hence, if the p-value is lower than 0.001, then the performance of the two planners is statistically different. *Note that when this information is given on the left side of the cell, the first planner named in the title of the cell performs better than the other compared planner.* The value under each cell in Table 6 is the number of problems solved by at least one planner.

Observation 7. *According to the results of the Wilcoxon test for the benchmark problems considered, in terms of CPU-time* Metric-LPG.s *performs consistently better than the other tested planners.*

6.3.3. Performance in terms of (only) plan quality

Fig. 14 shows a graphical representation of the overall performance of Metric-LPG.q with respect to Metric-FF (circles), MIPS (triangles) and SGPLAN5 (crosses) in terms of plan quality for our benchmark problems. For this analysis, we considered all the problems solved by *both* the compared planners. A circle above (below) the solid diagonal corresponds to a problem for which the plan computed by Metric-LPG.q is better (worse) than the plan computed by Metric-FF. The circles above the line labeled "2tB" (below the diagonal labeled "2tW") correspond to problems for which the plans computed by Metric-LPG.q are at least two times better (worse) than the ones computed by Metric-FF. Analogously for the triangles and the crosses. Since almost all the circles, triangles and crosses are

[17] For the unsolved problems, instead of infinity, we alternatively considered 600 seconds (i.e., the maximum CPU-time limit). By a confidence level of 99.9%, the qualitative results of the Wilcoxon test remain the same.

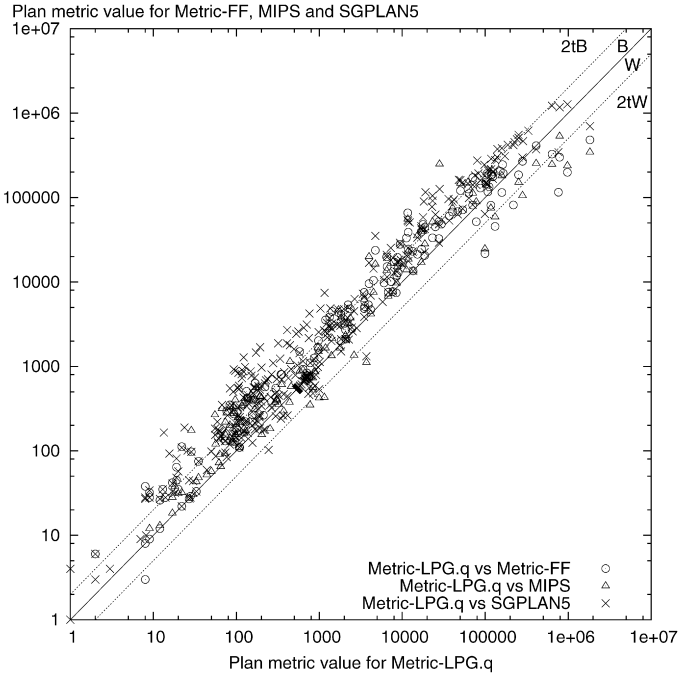


Fig. 14. Scatterplot comparing the performance of Metric-LPG.q versus Metric-FF, MIPS and SGPLAN5 in terms of plan quality for our benchmark problems. On the x-axis, we have the plan qualities of Metric-LPG.q (log scale); on the y-axis we have the plan qualities of the compared planners (log scale). Lower plan metric values correspond to better plans. "B/W" means better/worse plans; "2tB/2tW" means at least 2 times better/worse plans.

Table 7
Number of test problems for which the plans computed by Metric-LPG.q are worse (2nd and 3rd columns) and better (4th and 5th columns) than those computed by Metric-FF, MIPS and SGPLAN5 for our benchmark problems. "B/W" means better/worse plans; "2tB/2tW" means at least 2 times better/worse plans

Plan quality analysis	2tW (Zeno)	W (Zeno)	B	2tB
Metric-LPG.q vs Metric-FF	8 (7)	15 (13)	115	54
Metric-LPG.q vs MIPS	10 (10)	25 (24)	110	45
Metric-LPG.q vs SGPLAN5	4 (3)	24 (10)	409	170

above the main diagonal, this clearly indicates that the plans computed by Metric-LPG.q are generally better than those computed by any other compared planner.

Table 7 gives detailed information about the number of circles, triangles and crosses in each area of Fig. 14. For most of the tested problems, Metric-LPG.q computes better plans. Only a small percentage of the plans computed by Metric-LPG.q are worse than the plans computed by the compared planners. Moreover, most of these plans concern the largest problems in the `ZenoTravel` domain. We observed that for these problems the heuristic search neighborhood is very large, which excessively slows down each search step. For this reason, although the planner finds a solution, the given CPU-limit is too tight to derive additional plans with better quality.

As for the CPU-time analysis, we use the Wilcoxon test to show that the performance gaps in terms of plan quality are significant. The test procedure is similar to the one previously described for the CPU-time analysis, except that here we consider all the problems solved by both the compared planners, and we compute the differences between the plan qualities of the planners being compared. Table 8 gives the results of the Wilcoxon test about the plan quality for Metric-LPG.q and the other tested planners. The value under each cell is the number of problems solved by both the compared planners.

Observation 8. *According to the results of the Wilcoxon test for the benchmark problems considered,* Metric-LPG.q *computes plans that are consistently better than those computed by the other tested planners.*

Table 8
Results of the Wilcoxon test for the performance of Metric-LPG.q versus MIPS, Metric-FF and SGPLAN5 in terms of plan quality for our benchmark problems

Metric-LPG.q *vs* Metric-FF		Metric-LPG.q *vs* MIPS		Metric-LPG.q *vs* SGPLAN5	
-6.35		-5.31		-16.0	
< 0.001		< 0.001		< 0.001	
153		146		455	

6.3.4. Performance in terms of both CPU-time and plan quality

The previous experimental analysis show that, if we are only interested in having a solution plan as quickly as possible Metric-LPG.s performs very well; similarly, if we are only interested in having a solution as good as possible without considering the plan generation time (below a fixed limit of 10 CPU-minutes), Metric-LPG.q performs remarkably well.

In this section we use the Wilcoxon test to analyze the overall performance of our planner with respect to the other tested planners in terms of *both* CPU-time and plan quality. The tables in Fig. 15 give the results about the relative performance of: (1) Metric-LPG.s versus the other tested planners in terms of plan quality; (2) Metric-LPG.q versus the other tested planners in terms of CPU-time. For the analysis concerning CPU-time, the value under each cell is the number of problems solved by at least one planner; while for the analysis concerning plan quality, it is the number of problems solved by both the compared planners. Fig. 16 gives a graphical summary of the Wilcoxon results about the relative performance of Metric-LPG.s/q, MIPS, Metric-FF and SGPLAN5 in terms of both CPU-time and plan quality for our benchmark problems. A solid arrow from a planner A to a planner B (or to a cluster of planners B) indicates that the performance of A is statistically different from the performance of B (every planner in B), and that A performs better than B (every planner in B). A dashed arrow from A to B indicates that A is better than B with a confidence level slightly less than 99.9%.

Observation 9. *According to the Wilcoxon test for the benchmark problems considered,*

- Metric-LPG.s *performs consistently better than* SGPLAN5 *in terms of* both *CPU-time and plan quality*; Metric-LPG.q *performs consistently better than* MIPS *in terms of* both *CPU-time and plan quality*;
- Metric-LPG.s *performs consistently better than* Metric-FF *in terms of CPU-time, but not in terms of plan quality*; Metric-LPG.q *performs consistently better than* Metric-FF *in terms of plan quality, but not in terms of CPU-time.*

The mixed results in the comparison of Metric-LPG.s/q and Metric-FF indicates that there could exist an intermediate set of solutions computed by Metric-LPG that is consistently better than Metric-FF in terms of both CPU-time and plan quality. In fact, we observed that this is the case. For instance, we analyzed the set of the best quality plans generated for our benchmark problems within 10 CPU-seconds after Metric-LPG has found the first plan. According to the

CPU-Time Analysis							
Metric-LPG.q *vs* Metric-LPG.s		Metric-LPG.q *vs* Metric-FF		Metric-LPG.q *vs* MIPS		Metric-LPG.q *vs* SGPLAN5	
	-18.1		-0.75	-8.31			-7.56
	< 0.001		(0.455)	< 0.001			< 0.001
539		208		301		561	

Plan Quality Analysis							
Metric-LPG.s *vs* Metric-LPG.q *vs*		Metric-LPG.s *vs* Metric-FF		Metric-LPG.s *vs* MIPS		Metric-LPG.s *vs* SGPLAN5	
	-17.6		-1.10		-7.85	-4.56	
	< 0.001		(0.273)		< 0.001	< 0.001	
539		153		146		455	

Fig. 15. Results of the Wilcoxon test about the performance of Metric-LPG.s/q, MIPS, Metric-FF and SGPLAN5 in terms of CPU-time and plan quality for our benchmark problems.

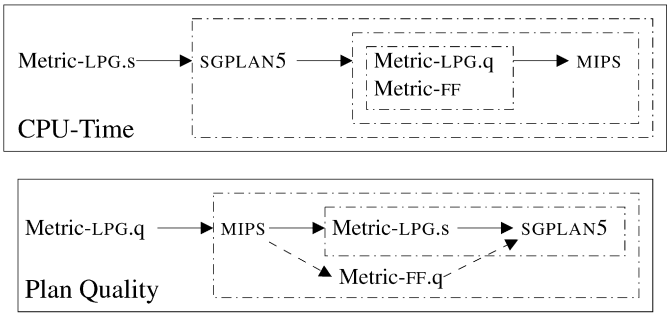


Fig. 16. Partial order of the performance of Metric-LPG.s/q, MIPS, Metric-FF and SGPLAN5 according to the Wilcoxon test for our benchmark problems.

Wilcoxon tests using this set of intermediate solutions, Metric-LPG performs consistently better than Metric-FF both in terms of CPU-time ($z = -3.35$, $p < 0.001$) and plan quality ($z = -4.05$, $p < 0.001$).

6.3.5. Performance for problems with tight resources

In order to compare Metric-LPG with MIPS, Metric-FF and SGPLAN for solving planning problems with tight resource availability constraints, we developed a collection of problems with varying resource availability for the PDDL formalization of the original Traveling Purchaser Problem (TPP) [40]. This analysis was inspired by a similar experiment recently conducted by Hoffmann and colleagues using a different domain [30].

TPP is a well-known generalization of the Traveling Salesman Problem and is NP-hard. We have a set of goods and a set of markets. Each market is provided with a limited amount of each type of goods at a known price. The problem consists in selecting a subset of markets such that a given demand of each type of goods can be purchased, minimizing the routing cost and the purchasing cost. TPP-like problems arise in several applications, and particularly in routing and scheduling contexts.

The PDDL formalization of TPP contains the following operators: `drive`, for moving a truck through the network of markets; `load` and `unload`, for loading/unloading goods into/from a truck; and `buy`, for purchasing goods at a market. In the PDDL operator specification file, we have a numerical fluent called "`money`" representing the amount of the available money. Since moving the truck from a market to another one and purchasing some goods consume money, the `drive` and `buy` operators contain the numerical effects

```
(decrease (money) (drive-cost ?from ?to)) and
(decrease (money) (price ?g ?m))
```

respectively, where `drive-cost` represents the cost for driving from location "`?from`" to location "`?to`", and `price` represents the price of goods "`?g`" at the market "`?m`".

We randomly generated 100 instances of the problem with 8 markets, 8 goods to buy (randomly placed at different locations) and 1 truck. In the PDDL problem specification, we set the initial value of `money` to *c times the minimal amount required to solve the problem* (we calculated the minimal amount of money by running a domain-specific planner that we developed for this particular domain). We considered each value of c in $\{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$, obtaining six sets of 100 problem instances each (only differing for the value of c). Note that when $c = 1$, solving each planning problem in the corresponding set requires the planner finding an optimal solution.

We analyzed the percentage of problems solved by Metric-LPG (speed version), Metric-FF, MIPS and SGPLAN with respect to a given CPU-time limit ranging from 10 milliseconds to 1000 seconds. (For this experiment we used version 4 of SGPLAN since, surprisingly, for these test problems SGPLAN4 works much better than SGPLAN5.) The results are given in Fig. 17. For CPU-time limits greater than 1 seconds, Metric-LPG always solves more problems than any other compared planner. The performance gap is greater for the problems with the tightest resource availability. In particular, within a CPU-time limit of 1000 seconds, for $c = 1$ Metric-LPG optimally solves 66 problems over 100, Metric-FF and SGPLAN only 2 and MIPS only 1. For $c = 1.1$, Metric-LPG works remarkably well, solving all problems, while the other compared planners solve only a small fraction of them.

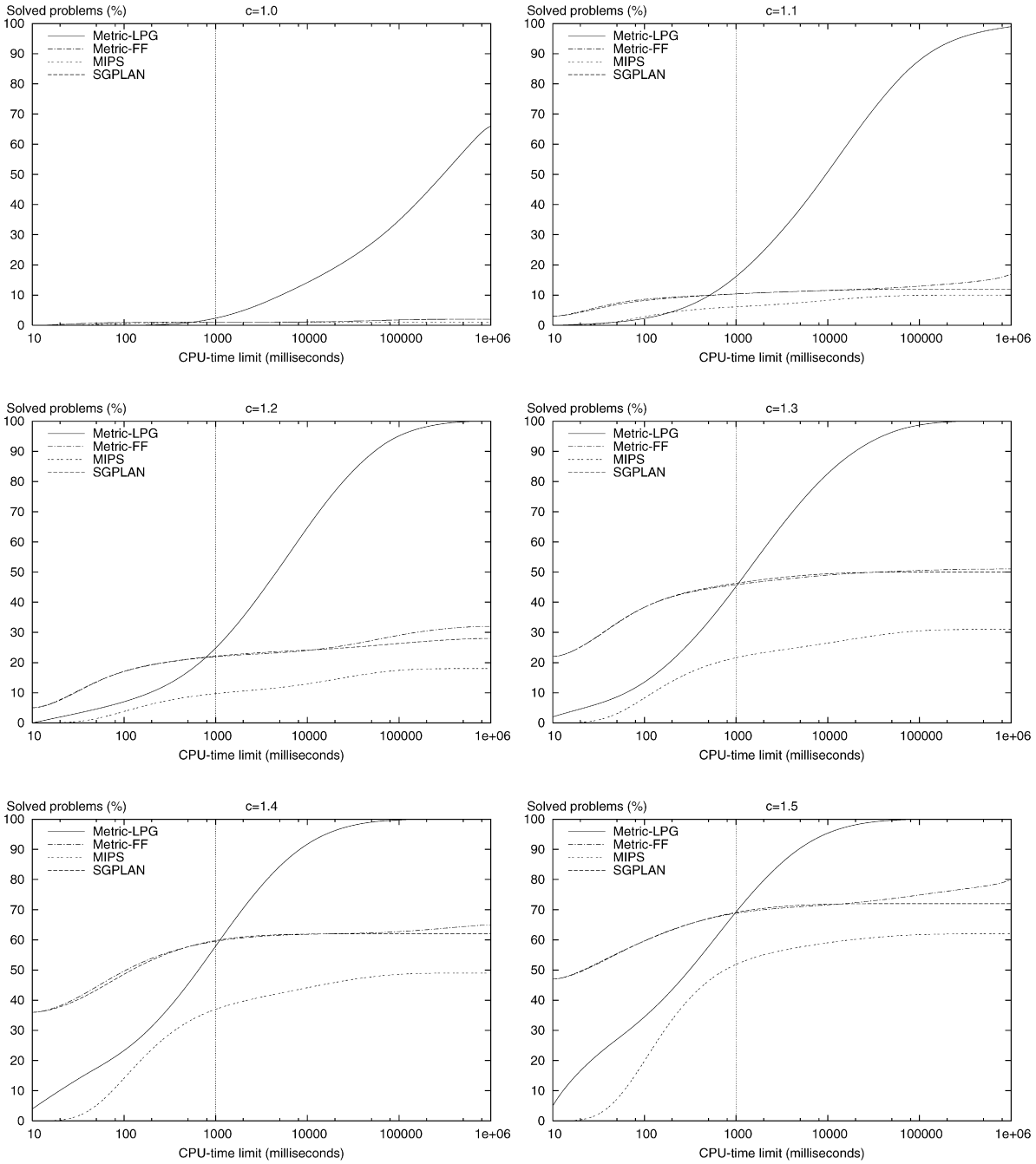


Fig. 17. Percentage of problems solved by Metric-LPG, Metric-FF, MIPS and SGPLAN with respect to an increasing CPU-time limit (ranging from 10 milliseconds to 1000 seconds) for the TPP problem [40]. Each plot corresponds to a different degree of tightness (c) for the available money. The CPU-time limit on the x-axis is in milliseconds (log scale).

Observation 10. *In a* PDDL *domain formalization of the Traveling Purchaser Problem* Metric-LPG *solves more problems with tight resources than the other tested satisficing planners.*

Hoffmann and colleagues obtained similar (although somewhat less detailed and differently presented) results for a variant of the Traveling Salesman Problem [30]. Their analysis show that for problem instances with tight resources,

our planner works better than all other tested satisficing planners. However, for problems where the resource tightness is extremely strong their optimal SAT-based planner solves more problems within 10,000 CPU-seconds.

7. Related work

In this section, first we describe the main differences with our previous related work [15,17]; then we briefly discuss other planning systems in the literature handling numerical information.

7.1. Previous work on LPG

The work presented in this paper adopts the general planning framework based on action graphs and local search, which is also investigated in [15,17]. The work in [17] addresses significantly different planning problems concerning temporal planning with durative actions and action scheduling constraints imposed by predictable exogenous events. In [17] we focus on efficient temporal reasoning integrated into the basic action graph representation, and heuristics for planning with the extended plan representation. The plan makespan is the only criterion considered for plan quality, and no technique for computing good quality plans is given.

In [15] we present an earlier version of our planner (here called LPG-IPC3). The work in [15] focuses on propositional and temporal planning, and includes only a short abstract description of preliminary techniques for handling numerical information implemented in LPG-IPC3. In this paper, we have given a comprehensive, detailed and more formal treatment of planning with numeric information for PDDL2.1 domains in the context of the general approach to planning with action graphs and local search. The work in this paper includes substantial new technical material and significantly extends our previous preliminary treatment of numerical fluents. In particular, we have given (for a more detailed discussion see [18]):

- *A new definition of the search neighborhood*. In order to deal with a numerical flaw in the current NA-graph (plan), a new action node or an existing action node can be added or removed, respectively, to/from the (earliest) flawed level of the graph *or to/from any preceding level* (provided that the added/removed node helps to resolve the flaw). In the previous definition, the addition or removal of an action for a numerical flaw could be done *only* at the flawed level. The revised neighborhood definition can lead to new neighborhood elements with better heuristic evaluations, but it can contain many elements to evaluate. Hence, we have defined the new *heuristic search neighborhood*, which contains only a subset of the elements in the revised search neighborhood. In Section 6.2 we have demonstrated the practical usefulness of this technique.
- *New evaluation techniques for the heuristic search neighborhood*. The relaxed-plan based techniques for the heuristic evaluation of the costs of adding/removing an action node have been improved by new techniques for computing the relaxed numerical plans. First, the goals of the relaxed problem are extended with (a) the supported *numerical* preconditions of the actions in the current plan that would become unsupported by performing the search step under evaluation; (b) all unsupported preconditions at the flawed level l under consideration (except when we are evaluating the removal of the action at the flawed level l). Secondly, the treatment of the numerical preconditions in the construction of the relaxed plan is more accurate: in LPG-IPC3, we used a much stronger relaxation assuming that only one action was sufficient to satisfy a numerical precondition. Finally, the choice of the (heuristic) best action for supporting a precondition p during the construction of the relaxed numerical plans exploits some heuristic numerical information which in LPG-IPC3 is completely ignored.
- *A new incremental method for computing good quality plans*. The techniques described in Section 4.5 for the incremental improvement of the plan quality are new. The version of our planner described in [15] does not assure that at each search restart the quality of the next plan generated by the search procedure is better than the previous ones. Moreover, the search initialization technique introduced and analyzed in this paper is completely new, and the experimental analysis in Section 6.2 shows that it works well. Finally, the previous version of our planner did not appropriately treat numerical planning problems where the plan metric is an expression to maximize.
- *A new extensive experimental analysis* in which: (1) we evaluate some of the proposed techniques in terms of their impact on the performance of our approach, and the effectiveness of our incremental plan quality optimization process; (2) we compare the performance of our planner with other state-of-the-art planners in terms of number

of problems solved with respect to different CPU-time limits, speed, plan quality, and ability of solving numerical problems when the availability of the involved resources has different degrees of tightness.

An experimental comparison of the new version of our planner and an improved version of LPG-IPC3 confirmed that the new techniques make the approach significantly more powerful.[18] The LPG-IPC3 like version solved 57% of the 644 test problems over 17 domains that we considered for our experimental analysis, while the new version solved 84% of them. In terms of speed, for every test domain except `Openstacks`, the new version of our planner showed a clear improvement (detailed results for each domain are given in [18]). The `Openstacks` problems for which the improved version of LPG-IPC3 is faster than Metric-LPG are solved by both planners in few milliseconds, and hence they are not significant for the comparison. Overall, Metric-LPG is faster in finding a solution for 86% of the 644 test problems solved by at least one of the two compared versions. More precisely, the new version is one or more orders of magnitude faster for *at least* 40.1% of the problems (this is a lower bound because here we consider the unsolved problems as if they were solved with a CPU-time equal to the exceeded search limit); two or more orders faster for at least 20% of the problems; and three or more orders for at least 6.12% of the problems. Metric-LPG is slightly slower for only 0.05% of the problems (these are all "easy" problems, most of which are solved in less than one second).

7.2. Other systems handling numerical information

In the planning literature, various approaches to (fully-automated) domain-independent planning supporting numerical information have been proposed [4,5,8,9,12,24,27,30,33,34,38,39,42,43,49]. In the following, we discuss those that are more closely related to our work.

Like in our approach, the search heuristics used in Hoffmann's Metric-FF [31] are based on relaxed plans, and the (relaxed) satisfaction of the numerical (sub)goals forming the relaxed plans relies on an estimate of the possible minimum and maximum values of the involved numerical fluents. The main difference concerns (a) the goals of the relaxed problem and (b) the heuristic method for choosing actions in the construction of the relaxed plan.

Concerning (a), in Metric-FF the goals are static (they are always the unsatisfied problem goals), while in Metric-LPG they are dynamic and can be completely different for each element in the search neighborhood. Moreover, the relaxed problem goals of Metric-LPG include the threats that the graph modification under consideration poses to the preconditions of the actions in the (real) plan. An example of a domain where handling the threats is very useful for Metric-LPG is the "flaw" version of `Umts` [29], where our planner recognizes that the special action `flaw` should not be included in the current plan, because it threatens a supported precondition that is satisfied in the initial state and cannot be re-established by any other domain action.

Concerning (b), we take account of the possible negative interferences between the actions in the relaxed plan and the actions in the (real) plan represented by the current NA-graph, which cannot be done in Metric-FF because of the significantly different search strategy and search spaces. Moreover, we consider the number of needed occurrences of the action under consideration for the relaxed plan, and we use different reachability information (*Num_acts*) to estimate the search cost of achieving a precondition p, which we think is more informative than Metric-FF information (Metric-FF considers the minimum level of the relaxed planning graph where p is supported, while we estimate the minimum number of *actions* needed to support p).

The MIPS system developed by Edelkamp [9] is based on an approach integrating model checking techniques into a domain-independent action planner. MIPS deals with numerical preconditions and goals by using relaxed planning-graph heuristic techniques similar to those developed for Metric-FF.

The SGPLAN planning system by Chen, Hsu and Wha [4,5] solves numerical planning problems by a powerful method decomposing the problem into a collection of subproblems, each of which is solved by running Metric-FF, appropriately modified to take account of the interferences with other subplans and to be incorporated into SGPLAN's architecture. For problems involving "producible resources" (i.e., problems with numerical fluents increased by an action without preconditions, or whose preconditions are always reachable), SGPLAN uses a special technique that includes at the beginning of the plan some actions producing a high quality of producible resources, making the satisfaction of the numerical conditions involving such resources much easier. This technique works especially well

[18] In the improved version of LPG-IPC3 we have fixed few implementation bugs and improved the parser; without these changes the improvements of Metric-LPG are even more significant.

in the Settlers domain, where SGPLAN generally performs better than our planner. For the problems where the default method of SGPLAN (IPC-4 version) fails, as an alternative planning technique, SGPLAN runs an old and less powerful version of our planner.

The SAPA planner by Do and Kambhampati [8] is based on a forward A* algorithm searching a space of time-stamped world states augmented with numerical fluents. The search states are estimated by a multi-criteria objective heuristic function considering both the makespan and the action costs. SAPA evaluates the possible successor search states by constructing relaxed plans for achieving the problem goals from such states, while Metric-LPG constructs relaxed plans for the unsupported precondition(s) of some plan action and possibly for the threats it determines (if the graph modification under consideration is an action node addition). In SAPA, the selection of an action during the construction of a relaxed plan is based on the impact that the action has on the makespan and execution cost of the relaxed plan. Differently from SAPA, the selection of an action for our relaxed plan does not explicitly consider plan quality and is aimed at minimizing the search cost/plan generation speed. Moreover, in the construction of the relaxed plan SAPA ignores the numerical preconditions of the actions in the plans and the numerical goals, which, on the contrary, are considered in the construction of our relaxed plans. Differently from our method, SAPA extends the execution cost of its relaxed plan by adding the cost of some extra actions that "balance the resource consumption in the relaxed plan". SAPA uses the (extended) total execution cost of the actions in the relaxed plan and its makespan for the heuristic evaluation of the candidate successor search states, without considering their search cost. Differently from SAPA, Metric-LPG evaluates the elements of the search neighborhood by combining their search costs and execution costs, which are both derived from the actions forming the relaxed plan. According to the results of IPC-3 [36], SAPA performs less efficiently than our planner.

While, in principle, Metric-FF and SAPA's relaxed plans could be used in our context, we think that our method for computing them is much more accurate for our search space and search strategy. On the other hand, Metric-LPG's construction of the relaxed plan can only partially be used in the other two planners. In particular, considering the plan threats in the action selection can only be done in our context.

Finally, Refanidis and Vlahavas developed MO-GRT [43], which uses a weighted A* algorithm and a multi-objective heuristic function over a weighted hierarchy of user-defined criteria. This function is more expressive than the objective functions supported by Metric-LPG and PDDL2.1. In our approach, we use different search techniques, and the execution costs associated with the actions are automatically derived from the plan metric expression specified in the planning problem description, which can be a function involving multiple optimization criteria.

8. Conclusions

In many realistic planning domains, numerical fluents are essential to adequately model continuous resources that can be consumed or produced by the plan actions, and that are involved in the action preconditions, action effects and plan quality measure. In general, a planner has to take the problem numerical information into account to guarantee the correctness of its plans, to generate plans of good or optimal quality according to the given plan metric, and to quickly find a valid plan by effective search heuristics. In the recent years, there has been considerable interest in planning with numerical information, and, since the introduction of numerical fluents in the PDDL language, several powerful systems supporting them have been proposed.

In this paper, we have presented an approach to satisficing planning in the context of a model for planning with numerical fluents and multi-criteria plan quality metrics based on the semantics of PDDL2.1 [11] (with some minor differences discussed in the paper). The approach consists of a new plan representation extending the linear action graph with numerical information and some new techniques for planning through local search using this representation. Local search methods and satisficing planning in general do not guarantee optimal solutions, that, however, for many planning problems are computationally too hard to find for any state-of-the-art planner. Instead, depending on the planning domain, the main aims of satisficing planning are quickly finding any valid plan (that subsequently might be improved) and finding a good quality valid plan within a given amount of computational resources. Our approach addresses both these perspectives by quickly generating a first solution, and then computing additional solutions that incrementally improve plan quality.

All our techniques are implemented in Metric-LPG. An extensive experimental analysis presented in this paper evaluates the practical importance of the main proposed techniques for the effectiveness of our approach, and indicates

that Metric-LPG performs quite well with respect to other state-of-the-art satisficing planners, especially in terms of plan quality.

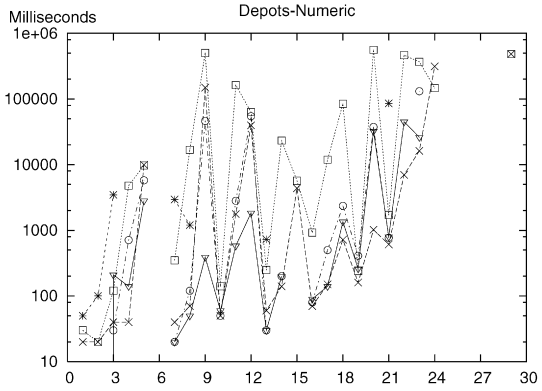
Future work includes extending our techniques to handle explicitly equality conditions involving numerical fluents, that currently Metric-LPG does not support. A more significant extension concerns adapting our methods to treat planning with soft preconditions and goals, which can be encoded into a particular form of numerical planning, and addressing over-subscription planning (e.g., [2,46]). Finally, we plan to investigate a problem decomposition method similar to the one developed for SGPLAN, using an appropriately modified version of Metric-LPG as subplanner.

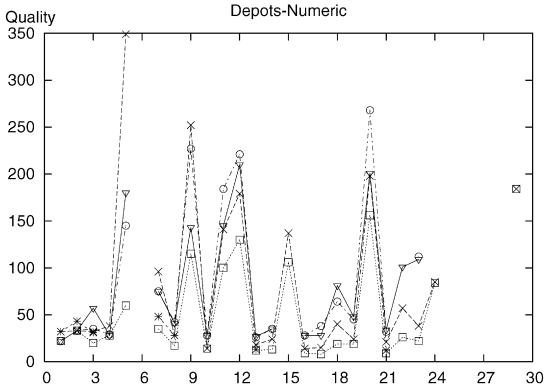
Acknowledgements

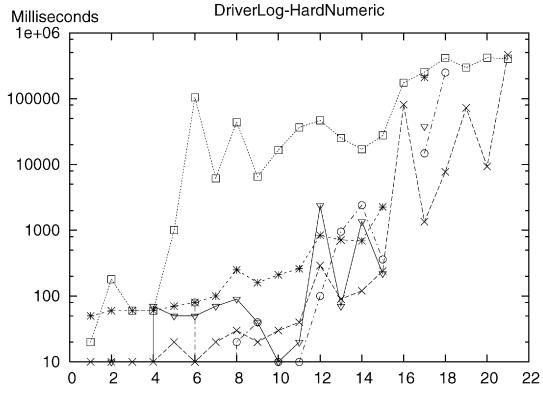
We would like to thank the anonymous reviewers for many helpful comments, and Maria Fox and Derek Long for some clarifications about the semantics of PDDL2.1. Metric-LPG has been implemented with the help of several undergraduate students at the University of Brescia. We would like to thank, in particular, Alberto Bettini, Marco Lazzaroni, Sergio Spinoni and Paolo Toninelli.

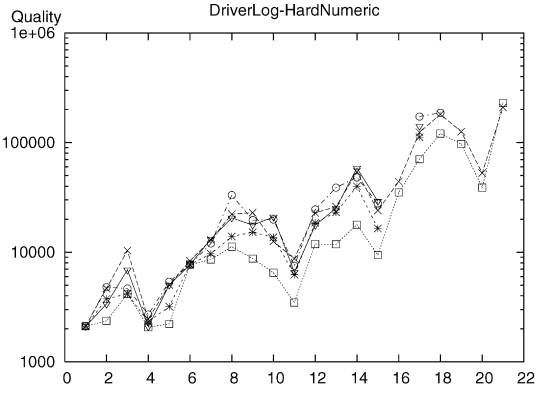
Appendix A. Performance of Metric-LPG and other planners

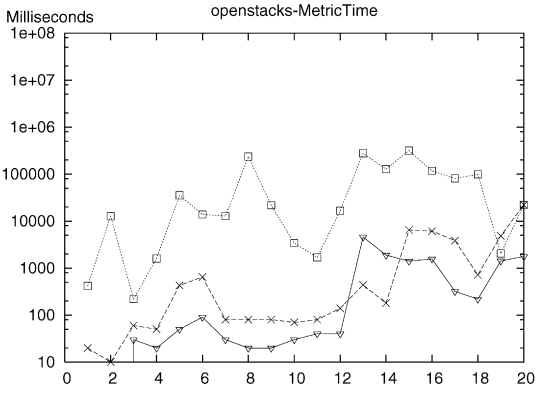
The following plots show the performance of Metric-LPG, Metric-FF, MIPS and SGPLAN5 for the benchmark problems in some of the 17 IPC metric and metric-time domains considered for our experimental analysis (all the other plots are available in [18]). In the main text of the paper, we give an overall evaluation for all domains based on scatter-plots and the Wilcoxon test. *In each plot, crosses indicate* Metric-LPG.s, *squares* Metric-LPG.q, *circles* Metric-FF, *stars* MIPS and *triangles* SGPLAN5. Notice that not every planner appears in every plot. This is the case because Metric-FF does not support temporal information, and because in some domains MIPS solves no problem.

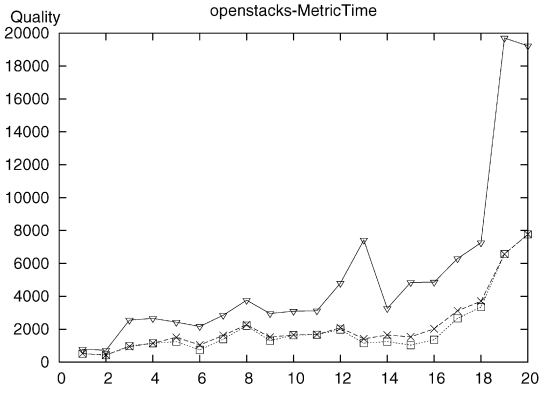


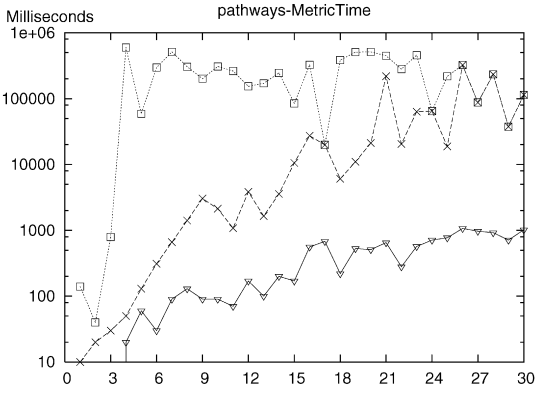


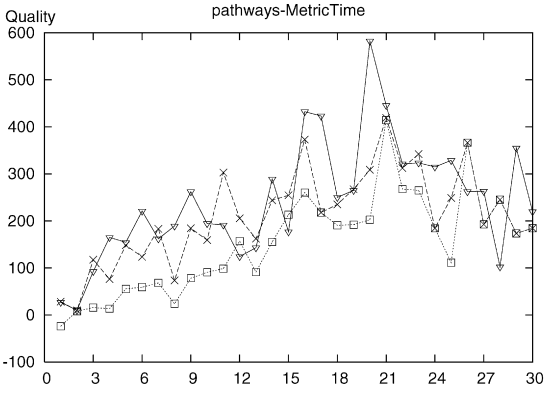


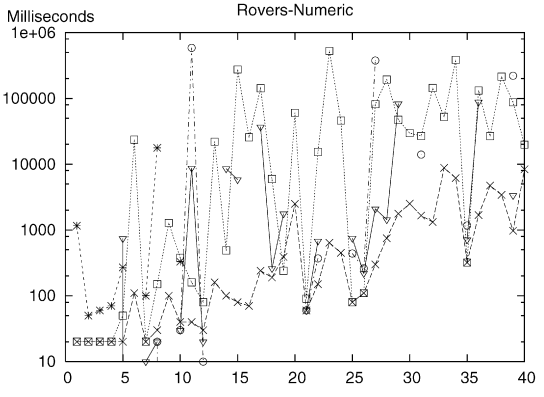


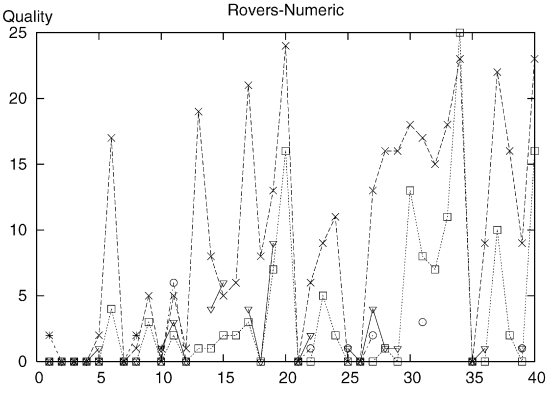

Milliseconds
openstacks-MetricTime
1e+08
1e+07
1e+06
100000
10000
1000
100
10
0 2 4 6 8 10 12 14 16 18 20

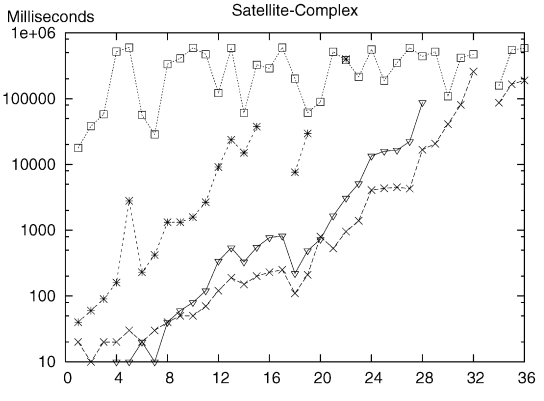

Quality
openstacks-MetricTime
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0 2 4 6 8 10 12 14 16 18 20

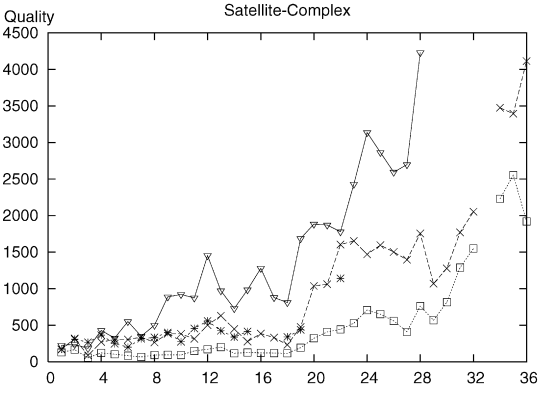

Milliseconds
pathways-MetricTime
1e+06
100000
10000
1000
100
10
0 3 6 9 12 15 18 21 24 27 30

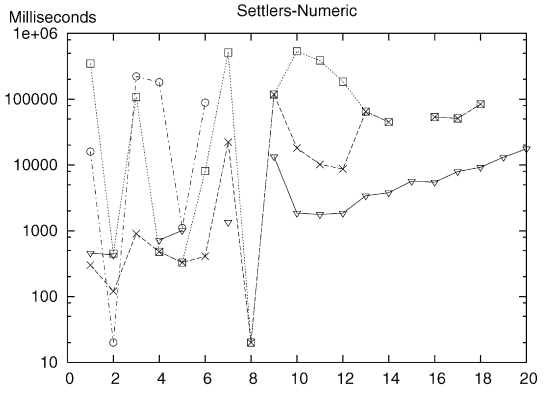

Quality
pathways-MetricTime
600
500
400
300
200
100
0
-100
0 3 6 9 12 15 18 21 24 27 30

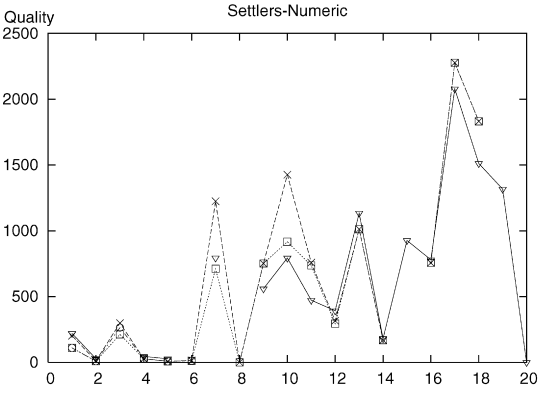

Milliseconds
Rovers-Numeric
1e+06
100000
10000
1000
100
10
0 5 10 15 20 25 30 35 40

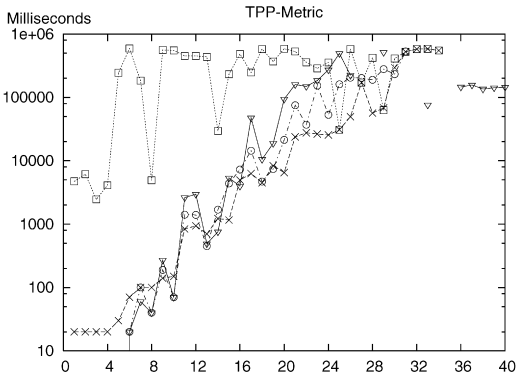

Quality
Rovers-Numeric
25
20
15
10
5
0
0 5 10 15 20 25 30 35 40

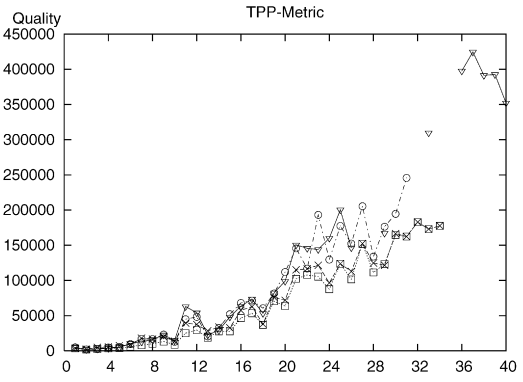

Milliseconds
Satellite-Complex
1e+06
100000
10000
1000
100
10
0 4 8 12 16 20 24 28 32 36

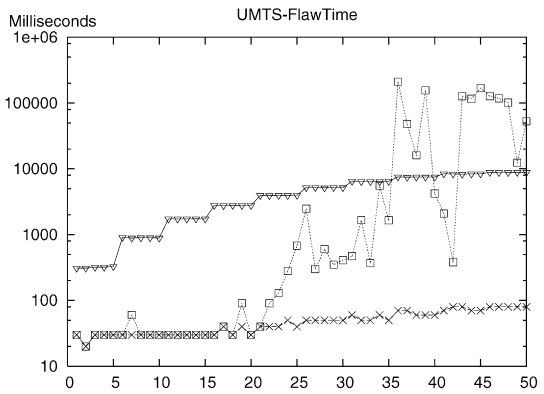

Quality
Satellite-Complex
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0 4 8 12 16 20 24 28 32 36

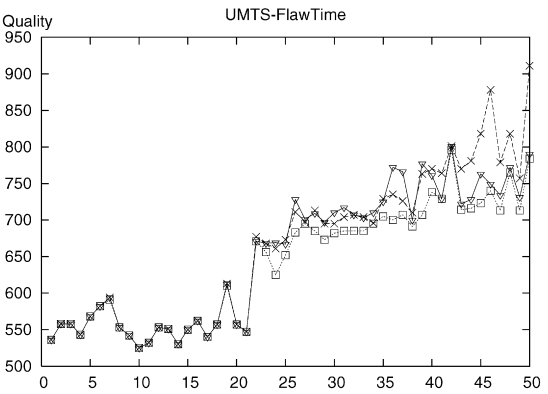

Settlers-Numeric
Milliseconds
1e+06
100000
10000
1000
100
10
0 2 4 6 8 10 12 14 16 18 20

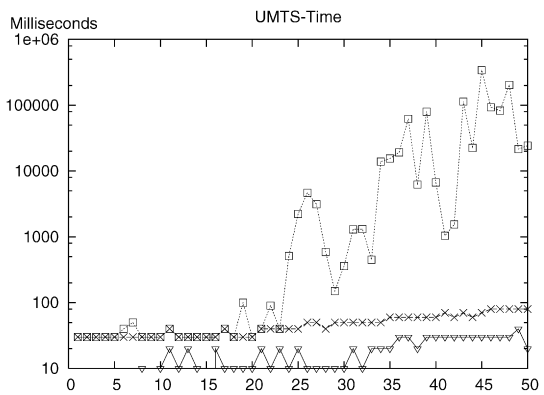

Settlers-Numeric
Quality
2500
2000
1500
1000
500
0
0 2 4 6 8 10 12 14 16 18 20

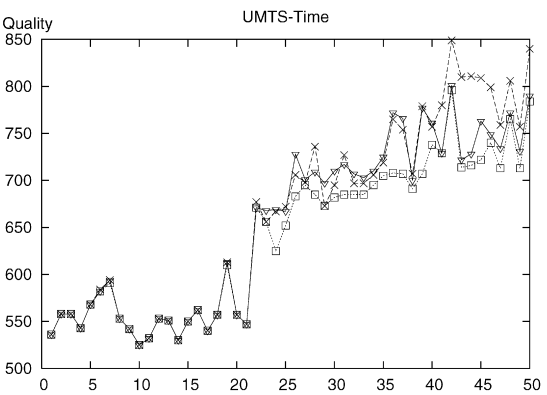

TPP-Metric
Milliseconds
1e+06
100000
10000
1000
100
10
0 4 8 12 16 20 24 28 32 36 40

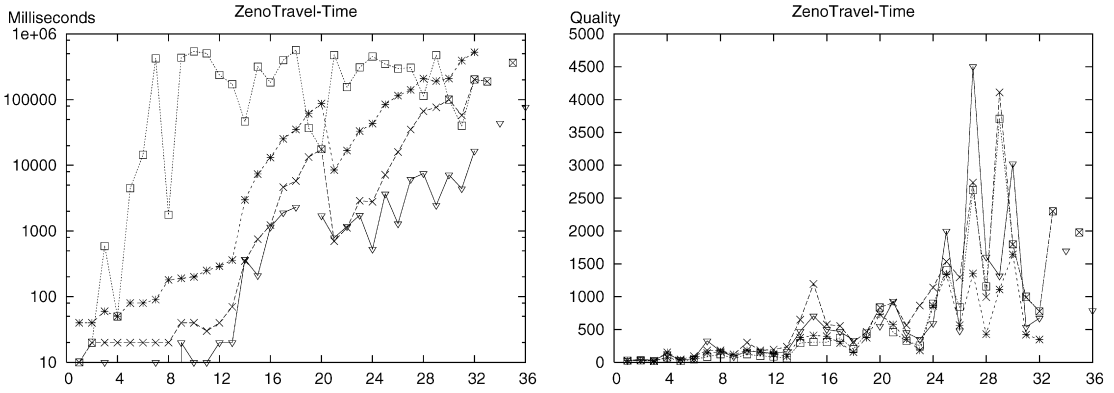

TPP-Metric
Quality
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
0 4 8 12 16 20 24 28 32 36 40


UMTS-FlawTime
Milliseconds
1e+06
100000
10000
1000
100
10
0 5 10 15 20 25 30 35 40 45 50


UMTS-FlawTime
Quality
950
900
850
800
750
700
650
600
550
500
0 5 10 15 20 25 30 35 40 45 50


UMTS-Time
Milliseconds
1e+06
100000
10000
1000
100
10
0 5 10 15 20 25 30 35 40 45 50


UMTS-Time
Quality
850
800
750
700
650
600
550
500
0 5 10 15 20 25 30 35 40 45 50



References

[1] F. Bacchus, M. Ady, Planning with resources and concurrency: A forward chaining approach, in: Proceedings 17th International Joint Conference on Artificial Intelligence (IJCAI-01), Seattle, WA, USA, 2001, pp. 417–424.
[2] J. Benton, M.B. Do, S. Kambhampati, Over-subscription planning with numeric goals, in: Proceedings 19th International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland, 2005, pp. 1207–1214.
[3] A. Blum, M.L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1997) 281–300.
[4] Y. Chen, C. Hsu, B. Wah, Temporal planning using subgoal partitioning and resolution in SGPlan, Journal of Artificial Intelligence Research (JAIR) 26 (2006) 323–369.
[5] Y. Chen, C. Hsu, W. Wha, SGPlan: Subgoal partitioning and resolution in planning, in: Abstract Booklet of the Competing Planners of ICAPS-04, 2004, pp. 30–32.
[6] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1991) 61–95.
[7] Y. Dimopoulos, A. Gerevini, P. Haslum, A. Saetti, The benchmark domains of the deterministic part of IPC-5, in: Abstract Booklet of the Competing Planners of ICAPS-06, 2006, pp. 14–19.
[8] M. Do, S. Kambhampati, Sapa: A multi-objective metric temporal planner, Journal of Artificial Intelligence Research (JAIR) 20 (2003) 155–194.
[9] S. Edelkamp, Taming numbers and duration in the model checking integrated planning system, Journal of Artificial Intelligence Research (JAIR) 20 (2003) 61–124.
[10] M. Fox, D. Long, Utilizing automatically inferred invariants in graph construction and search, in: Proceedings 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00), Breckenridge, CO, USA, 2000, pp. 102–111.
[11] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, Journal of Artificial Intelligence Research (JAIR) 20 (2003) 61–124.
[12] A. Garrido, D. Long, Planning with numeric variables in multi-objective planning, in: Proceedings 16th European Conference on Artificial Intelligence (ECAI-04), Valencia, Spain, 2004, pp. 662–666.
[13] A. Gerevini, D. Long, Plan constraints and preferences in PDDL3, Technical Report 2005-08-47, Università degli Studi di Brescia, Dipartimento di Elettronica per l'Automazione, 2005.
[14] A. Gerevini, A. Saetti, I. Serina, On managing temporal information for handling durative actions in LPG, in: AI*IA 2003: Advances in Artificial Intelligence, Springer-Verlag, Berlin, Heidelberg, New York, 2003, pp. 91–104.
[15] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action graphs, Journal of Artificial Intelligence Research (JAIR) 20 (2003) 239–290.
[16] A. Gerevini, A. Saetti, I. Serina, An empirical analysis of some heuristic features for local search in LPG, in: Proceedings 14th International Conference on Automated Planning and Scheduling (ICAPS-04), Whistler, Canada, 2004, pp. 171–180.
[17] A. Gerevini, A. Saetti, I. Serina, An approach to temporal planning and scheduling in domains with predictable exogenous events, Journal of Artificial Intelligence Research (JAIR) 25 (2006) 187–231.
[18] A. Gerevini, A. Saetti, I. Serina, An approach to efficient planning with numerical fluents and multi-criteria plan quality (preliminary report), Technical Report 2007-01-53, Università degli Studi di Brescia, Dip. di Elettronica per l'Automazione, 2007.
[19] A. Gerevini, I. Serina, Fast planning through greedy action graphs, in: Proceedings 16th National Conference on Artificial Intelligence (AAAI-99), Orlando, FL, USA, 1999, pp. 503–510.
[20] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL—the planning domain definition language, Technical Report CVC TR98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
[21] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2003.
[22] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, MA, USA, 1997.
[23] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Boston, MA, USA, 1989.
[24] P. Haslum, H. Geffner, Heuristic planning with time and resources, in: Proceedings 6th European Conference on Planning (ECP-01), Toledo, Spain, 2001.

[25] M. Helmert, Decidability and undecidability results for planning with numerical state variables, in: Proceedings 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02), Toulouse, France, 2002, pp. 303–312.
[26] M. Helmert, The Fast Downward planning system, Journal of Artificial Intelligence Research (JAIR) 26 (2006) 191–246.
[27] J. Hoffmann, The Metric-FF planning system: Translating "ignoring delete lists", to numeric state variables, Journal of Artificial Intelligence Research (JAIR) 20 (2003) 291–341.
[28] J. Hoffmann, S. Edelkamp, The deterministic part of IPC-4: An overview, Journal of Artificial Intelligence Research (JAIR) 24 (2005) 519–579.
[29] J. Hoffmann, S. Edelkamp, S. Thièbaux, R. Englert, F. Liporace, S. Trueg, Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4, Journal of Artificial Intelligence Research (JAIR) 26 (2006) 453–541.
[30] J. Hoffmann, H. Kautz, C. Gomes, B. Selman, SAT encodings of state-space reachability problems in numeric domains, in: Proceedings 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, 2007.
[31] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research (JAIR) 14 (2001) 253–302.
[32] C. Hsu, B.W. Wah, R. Huang, Y. Chen, New features in SGPlan for handling preferences and constraints in PDDL3.0, in: Abstract Booklet of the competing planners of IPC-5, 2006, pp. 39–41.
[33] H. Kautz, J.P. Walser, State-space planning by integer optimization, in: Proceedings 16th National Conference on Artificial Intelligence (AAAI-98), Madison, WI, USA, 1998, pp. 526–533.
[34] J. Koehler, Planning under resource constraints, in: Proceedings 13th European Conference on Artificial Intelligence (ECAI-98), Brighton, UK, 1998, pp. 489–493.
[35] J. Koehler, B. Nebel, J. Hoffmann, Y. Dimopoulos. Extending planning graphs to an ADL subset, Technical Report 88, Institut für Informatik, Freiburg, Germany, 1997.
[36] D. Long, M. Fox, The 3rd international planning competition: Results and analysis, Journal of Artificial Intelligence Research (JAIR) 20 (2003) 1–59.
[37] D. Nau, Y. Cao, A. Lotem, H. Muñoz Avila, SHOP: Simple hierarchical ordered planner, in: Proceedings 16th International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999, pp. 968–973.
[38] J.S. Penberthy, D.S. Weld, Temporal planning with continuous change, in: Proceedings 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, USA, 1994, pp. 1010–1015.
[39] S.J. Penberthy, Planning with continuous change, PhD thesis, University of Washington, 1993. Available as technical report UW-CSE-93-12-01.
[40] T. Ramesh, Traveling purchaser problem, Opsearch 18 (1981) 78–91.
[41] I. Refanidis, I. Vlahavas, GRT: A domain independent heuristic for STRIPS worlds based on greedy regression tables, in: Proceedings 5th European Conference on Planning (ECP-99), Durham, UK, 1999, pp. 346–358.
[42] I. Refanidis, I. Vlahavas, Heuristic planning with resources, in: Proceedings 14th European Conference on Artificial Intelligence (ECAI-00), Berlin, Germany, 2000, pp. 521–525.
[43] I. Refanidis, I. Vlahavas, Multiobjective heuristic state-space planning, Artificial Intelligence Journal 145 (1–2) (2003) 1–32.
[44] B. Selman, H.A. Kautz, B. Cohen, Noise strategies for improving local search, in: Proceedings 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, USA, 1994, pp. 337–343.
[45] A.H. Simon, Models of Man, John Wiley & Sons Inc., New York, USA, 1957.
[46] D. Smith, Choosing objectives in over-subscription planning, in: Proceedings 14th International Conference on Automated Planning and Scheduling (ICAPS-04), Whistler, Canada, 2004.
[47] F. Wilcoxon, R.A. Wilcox, Some Rapid Approximate Statistical Procedures, Lederle Laboratories, Pearl River, New York, USA, 1964.
[48] M. Williamson, S. Hanks, Optimal planning with a goal-directed utility model, in: Proceedings 2nd International Conference on Artificial Intelligence Planning and Scheduling (AIPS-94), Chicago, IL, USA, 1994, pp. 176–181.
[49] S. Wolfman, D. Weld, The LPSAT system and its applications to resource planning, in: Proceedings 16th International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999.