Planning with Numerical Expressions in LPG

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

Abstract. We present some techniques for handling planning problems with numerical expressions that can be specified using the standard planning language PDDL. These techniques are implemented in LPG, a fully-automated planner based on local search that was awarded at the third international planning competition (2002). First, we present a plan representation for handling numerical expressions called Numerical Action Graph (NA-graph). Then, we propose some extensions of LPG to guide a search process where the search states are NA-graphs. Finally, we give some experimental results showing that our techniques are very effective in terms of CPU-time or plan quality, and that they significantly improve the previous version of the planner.

1 Introduction

Local search is a powerful method for domain-independent planning, as demonstrated by two planners awarded at the second and third international planning competitions (IPC): FF [6] and LPG [5].

In this paper we present some extensions of LPG for handling planning problems involving numerical expressions specified with the standard planning language PDDL [3, 8]. In PDDL, numerical expressions are constructed using *primitive numerical expressions* and arithmetic operators. Numerical action preconditions use *comparison operators* (such as $<$, $\leq$, $=$, $\geq$, and $>$) involving pairs of numerical expressions. Numerical action effects use *assignment operators* (such as `assign`, `increase`, `decrease`, `scale-up`, and `scale-down`) for updating the value of some primitive numerical expression. Primitive numerical expressions are functions involving domain objects (e.g., `fuel(A1)`, whose value represents the amount of fuel available for `A1`).

The paper has three main contributions: (i) we present a plan representation for handling numerical expressions, that we call *Numerical Action Graph* (*NA-graph*); (ii) we propose new heuristic functions guiding a local search process where the search states are NA-graphs; (iii) we experimentally evaluate the proposed techniques using the numerical test problems of the 3rd IPC.

The results of the 3rd IPC showed that LPG is generally an efficient planner [7]. However, often the metric version of FF solved the numerical test problems more quickly than the version of LPG that took part in the competition (we will call such a version LPG-IPC3). LPG-IPC3 does not include the neighborhood evaluation functions presented in this paper. These new functions give a much more accurate estimate of the quality of the elements in the neighborhood, in terms of search cost to reach a valid plan, or execution cost of the plan under construction.

The experimental analysis in this paper show that the new version of LPG solves more than 93.2% of the numerical test problems of the 3rd IPC, while LPG-IPC3 solves only 59.9% of them, and Metric-FF 68.5% of them. Moreover, in 40% of the problem tested, the CPU-time required by the new version of LPG is at least one order of magnitude less than the CPU-time required by LPG-IPC3. The experimental results show also that the new version of LPG generates plans that generally have better quality than those produced by Metric-FF.

Section 2 introduces NA-graphs; Section 3 describes the basic local search procedure, the new search neighborhood, the heuristic evaluation of the search neighborhood, and a revised strategy to restart the search for improving plan quality; Section 4 gives the results of our experimental analysis using the numerical domain variants of the 3rd IPC; finally, Section 5 gives the conclusions.

2 Numerical Action Graph

In this section, we present our plan representation to handle numerical domains, which is an extension of the *action graph* [4], a particular subgraph of the well-known planning graph representation [1].

A **numerical action graph** (NA-graph) $\mathcal{A}$ is a directed acyclic leveled graph alternating a *fact level* and an *action level*. Fact levels contain *propositional nodes* and *numerical nodes*, labelled with propositions and numerical expressions, respectively. Numerical nodes are of two types: *numerical precondition nodes*, labelled with numerical comparisons, and *numerical fluent nodes*, labelled with primitive numerical expressions. Each numerical fluent node labelled x at a level l has a real value associated with it, that we denote with $NumVal(x, l)$. $NumVal(x, l)$ represents the value of the numerical primitive expression x at the world state corresponding to level l of $\mathcal{A}$. The value $NumVal(x, l)$ of a numerical fluent node x at level l (with $l > 1$) is derived from $NumVal(x, l-1)$ and the numerical effects of the action at level $l-1$ (if any). $NumVal(x, 1)$ is the value of x defined in the initial state of the planning problem.

Each action level contains one *action node* labelled with the name of a domain action, and any number of *no-op nodes* (defined as in [1]). Any action node labelled a at a level l is connected by (i) *precondition edges* to the propositional/numerical nodes at level l representing the preconditions of a, and (ii) by *effect edges* to the propositional/fluent nodes at level $l+1$ representing the effects of a. Each effect edge to a numerical fluent node is labelled with an assignment operator. Finally, if a is an action node of $\mathcal{A}$, then the precondition of a, the effect nodes of a, and the involved numerical fluent nodes are in $\mathcal{A}$, together with the edges connecting them to a

We assume that the problem goals are the preconditions of a special action a_{end}; while the initial facts and initial numerical fluent values are determined by the effects of a special action a_{start}.

Figure 1 gives an example of NA-graph for a simple logistics problem with numerical expressions (one airplane, `A1`, one package,

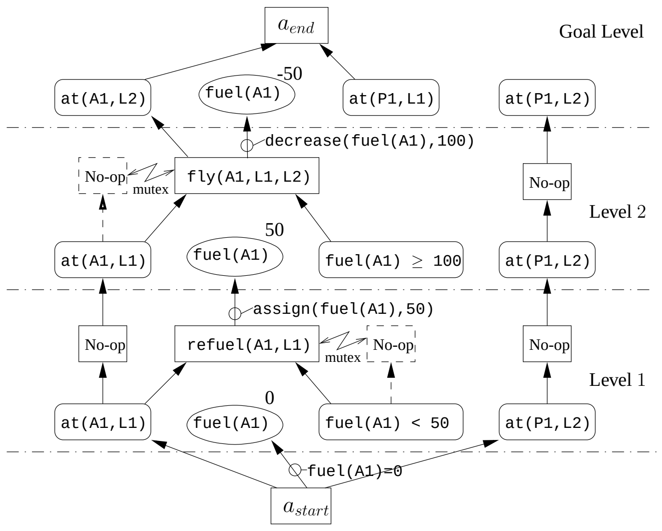

Figure 1. NA-graph for a simple logistics problem with numerical expressions. Numerical fluent nodes are marked with the corresponding values. `fly(A1,L1,L2)` blocks the propagation of `at(A1,L1)` at level 2, while `refuel(A1,L1)` blocks the propagation of `fuel(A1)<50` at level 1.

`P1`, and two locations, `L1` and `L2`). The value of `fuel(A1)` is 0 at level 1, 50 at level 2 (because of the effect `assign(fuel(A1),50)` of the action `refuel(A1,L1)`), and it is -50 at the goal level (because of the effect `decrease(fuel(A1),100)` of the action `fly(A1,L1,L2)`).

A numerical action graph can contain some *inconsistencies*, i.e., an action with a precondition node that is not *supported*. A numerical action graph without inconsistencies represents a valid plan, and it is called *solution graph*. A propositional precondition node q at a level l of a NA-graph $\mathcal{A}$ is supported if there is an action node (or a no-op node) at level $l-1$ of $\mathcal{A}$ connected to q by an effect edge. A numerical precondition node at a level l is supported if the corresponding numerical comparison is satisfied according to the values of the numerical fluent nodes at level l. For example, the numerical precondition node `fuel(A1)<50` at level 1 in Figure 1 is supported because the value assigned to `fuel(A1)` at level 1 is 0; on the contrary, `fuel(A1)`$\geq$`100` at level 2 is not supported, because of the action `refuel(A1,L1)` at level 1 assigning 50 to `fuel(A1)`.

The definition of NA-graph can be refined by including the (automatic) propagation of supported propositional nodes to the next levels of the graph through the corresponding no-ops, until there is an interfering action *blocking* the propagation, or the goal level has been reached. A similar propagation can be done also for supported numerical precondition nodes (for an example, see Figure 1).

In the rest of the paper we will use the following notion of *numerical state*.

Definition (Numerical State) *A* numerical state *is a pair* $\langle I, N\rangle$, *where* I *is a set of propositional facts, and* N *is an assignment of real values to the domain numerical fluents.*

Each level l of a NA-graph $\mathcal{A}$ identifies a particular numerical state, $\langle I_l, N_l\rangle$, obtained by applying to the initial numerical state of the planning problem the actions in $\mathcal{A}$ up to level $l-1$, ordered according to the corresponding level.

3 Local Search in the Space of NA-graphs

In this section, we present some techniques for searching in the space of NA-graphs that are implemented in the new version of LPG.

3.1 Background: Walkplan

The general scheme for searching a solution graph (a final state of the search) consists of a local search process in the space of all NA-graphs of the planning problem, starting from an initial NA-graph containing only a_{start} and a_{end}.

Each basic search step selects an inconsistency σ in the current NA-graph $\mathcal{A}$ and identifies the *neighborhood* $N(\sigma, \mathcal{A})$ of $\mathcal{A}$ for σ, i.e., the set of the NA-graphs obtained from $\mathcal{A}$ by applying a graph modification that resolves σ, or that simply *helps* to resolve σ. The elements of the neighborhood are weighed according to an *evaluation function* estimating their quality, and an element with the best quality is then considered as the next possible NA-graph (search state). The quality of a NA-graph depends on the number of the inconsistencies it contains, the estimated number of the search steps required to resolve them, and the overall execution or temporal cost (depending on the plan metric specified) of the represented plan.

The search strategy used by LPG is Walkplan, a method similar to the well-known procedure Walksat for solving propositional satisfiability problems [9]. According to Walkplan, the best element in the neighborhood is the NA-graph which has the *lowest decrease of quality* with respect to the current NA-graph, i.e., it does not consider possible improvements. This strategy uses a *noise parameter* p. Given a NA-graph $\mathcal{A}$ and an inconsistency σ, if there is a modification for σ that does not decrease the quality of $\mathcal{A}$, then the corresponding NA-graph is chosen as the next search state; otherwise, with probability p one of the graphs in $N(\sigma, \mathcal{A})$ is chosen randomly, and with probability $1-p$ the next NA-graph is chosen according to the minimum value of the evaluation function.

3.2 Neighborhood and Heuristics for NA-graphs

The search neighborhood $N(\sigma, \mathcal{A})$ of a NA-graph $\mathcal{A}$ for the selected inconsistency σ is the set of NA-graphs that can be derived from $\mathcal{A}$ by removing the action node with precondition σ or adding an action node that is *helpful* for σ. When we add an action node at a level l, the nodes and edges at each level $l' \geq l$ are shifted one level forward. Similarly, when we remove an action node, the NA-graph is "shrunk" by one level.

Definition (Helpful Action Node) *Given an unsupported precondition node* σ *at a level* l *of the current NA-graph* $\mathcal{A}$, *if* σ *is a propositional precondition, we say that* a *is a* helpful action node *for* σ *if its insertion into* $\mathcal{A}$ *at a level* $i \leq l$ *makes* σ *supported; if* σ *is a numerical precondition node labelled (*exp_1*,comp,*exp_2*), we say that* a *is a* helpful action node *for* σ *if its insertion into* $\mathcal{A}$ *at a level* $i \leq l$ *decreases the gap between the values of* exp_1 *and* exp_2 *at level* l *of* $\mathcal{A}$ *with respect to comp.*[1]

For example, consider the unsupported precondition (σ) `fuel(A1)`$\geq$`100` at level 2 of Figure 1. An action node with the numerical effect `increase(fuel(A1),50)` that is added at level 2 is helpful for σ, while such an action is not helpful if it is added at level 1 (because of the effect `assign(fuel(A1),50)` of `refuel(A1,L1)`).

The addition/removal of an action node a at level l of $\mathcal{A}$ may require a revision of the values associated with the numerical fluents at the next levels that are influenced by the numerical effects of a. Essentially, the algorithm for updating such values performs a forward

[1] This definition is similar to the one formulated by Hoffmann in [6], except that our approach handles both linear and non-linear numerical expressions.

EvalAdd(a)

Input: An action node a that does not belong to the current NA-graph.
Output: A set of pairs of type *(action, number of occurrences)*.

```
1. $l = Level(a)$;
2. $I_l \leftarrow SupportedFacts(l)$;
3. $N_l \leftarrow \{NumVal(x,l) \mid x$ is a numerical fluent$\}$;
4. $Rplan \leftarrow$ RelaxedNumplan($Pre(a), \langle I_l, N_l\rangle, \emptyset$);
5. $I_l^+ \leftarrow I_l \cup Add(a) - Threats(a)$;
6. $N_l^+ \leftarrow$ UpdateNumVal($N_l, a$);
7. $Rplan \leftarrow$ RelaxedNumplan($Threats(a), \langle I_l^+, N_l^+\rangle, Rplan$);
8. $Rplan \leftarrow Rplan \cup \{(a, Times(a,g))\}$;
9. return $Rplan$.
```

EvalDel(a)

Input: An action node a that belongs to the current NA-graph.
Output: A set of pairs of type *(action,occurrences)*.

```
1. $l = Level(a)$;
2. $I_l \leftarrow SupportedFacts(l)$;
3. $N_l \leftarrow \{NumVal(x,l) \mid x$ is a numerical fluent$\}$;
4. $Rplan \leftarrow$ RelaxedNumplan($UnsupFacts(a), \langle I_l, N_l\rangle, \emptyset$);
5. return $Rplan$.
```

Figure 2. Algorithms estimating the addition/removal of an action node a.

propagation starting from the effects of a, and updating level by level the numerical fluents from $l+1$ up to the goal level.

The elements of the neighborhood are evaluated according to an *action evaluation function* E estimating the cost of adding ($E(a)^i$) or removing ($E(a)^r$) an action node a. For STRIPS domains extended with numerical expressions, E consists of two weighed terms, evaluating the search cost and the quality of the current partial plan:

$$E(a)^i = \alpha \cdot SearchCost(a)^i \pm \beta \cdot ExecCost(a)^i,$$
$$E(a)^r = \alpha \cdot SearchCost(a)^r \pm \beta \cdot ExecCost(a)^r.$$

The first term of E estimates the increase of the number of search steps needed to reach a solution graph; the second estimates the increase of the plan execution cost. The coefficients of these terms, which are automatically set by our planner, are used to normalize them, and to weigh their relative importance. The sign of the second term depends from the plan metric expression (more details on this in Section 3.3).

Suppose that we are evaluating the addition of a at level l of the current NA-graph $\mathcal{A}$. The terms of E are heuristically estimated by computing a relaxed plan π. π consists of an estimated minimal set of actions to achieve the unsupported preconditions of a and the set Σ of the preconditions of other actions in $\mathcal{A}$ that would become unsupported by adding a (because it would block the no-op propagation currently used to support such preconditions).[2] π is relaxed in the sense that (i) it does not consider the negative interference with other actions in the relaxed plan; (ii) the possible minimum and maximum values of the involved numerical expressions are *monotonically* computed according to the actions in the relaxed plan; (iii) the comparisons of the numerical action preconditions are evaluated in a relaxed way, using the estimated min and max values of the involved numerical expressions. By monotonic computation of the min/max values of a numerical fluent x we mean the following: if an action in π increases x, we increase the max of x; if an action in π decreases x, we decrease the min of x.

The relaxed subplan of π for the preconditions of a is computed from the numerical state $\langle I_l, N_l\rangle$. The relaxed subplan of π for achieving Σ is computed from $\langle I_l, N_l\rangle$ modified by applying the effects of a. This subplan can reuse any action a' in the other relaxed subplan π previously computed for the preconditions of a. Note that, in order to support a numerical precondition, it can be necessary to use more than one action. For instance, if we have $x = 0$ in N_l, and

[2] Note that in LPG-IPC3 Σ contains only propositional preconditions, while here Σ can contain numerical preconditions as well.

RelaxedNumplan($G, \langle I_l, N_l\rangle, A$)

Input: A set of goals G, an initial numerical state $\langle I, N\rangle$, a set A of pairs (a,t);
Output: A set of pairs $\{(a,t)\}$ estimating the minimal set of actions with the relative number of occurrences that are required for achieving G .

```
1. $G \leftarrow G - I$; $F \leftarrow I$; $Acts \leftarrow A$;
2. forall $g \in G$ do
3.     $F \leftarrow \bigcup_{a \in Acts} Add(a)$;
4.     $\langle V_{min}, V_{max}\rangle$ = ComputeMinMax($N, Acts$);
5.     if $g \notin F$ and $g$ is not satisfied using $V_{min}$ and $V_{max}$ then
6.         $(b, tb) \leftarrow$ ChooseAction($g$);
7.         $Rplan \leftarrow$ RelaxedNumplan($Pre(b), \langle I, N\rangle, Acts$);
8.         forall $(a,t) \in Rplan$ such that $a = b$ do
9.             $Rplan \leftarrow Rplan - (a,t)$;
10.            $tb \leftarrow tb + t$;
11.        $Acts \leftarrow Rplan \cup \{(b, tb)\}$;
12. return $Acts$.
```

Figure 3. Algorithm for computing a relaxed plan achieving a set of action preconditions from the numerical state $\langle I, N\rangle$.

the only action modifying x increases it by 10 units, then to support `x > 25` we need three of such actions. In the following, we indicate t occurrences of an action a with the pair (a,t).

The actions in π are used to define a heuristic estimate of the additional search cost that would be introduced by the new action a ($SearchCost(a)$). This estimate also takes account of the number of supported preconditions that would become unsupported by adding the actions in π to $\mathcal{A}$ (because of their negative or numerical effects). The set of these subverted preconditions is denoted by $Threats(a)$. For example, if a has the numerical effect `decrease(x,50)` and $NumVal(x,l) = 120$, then the supported numerical precondition `x < 100` at level l becomes unsupported when adding a at level l. Note that LPG-IPC3 does not include numerical preconditions in $Threats(a)$.

$ExecCost(a)$ is an estimate of the additional execution cost that would be required to satisfy the preconditions of a, and it is derived by summing the execution cost of each a' in π ($Cost(a')$), multiplied by the corresponding number of occurrences.[3] More formally,

$$SearchCost(a)^i = \sum_{t \text{ s.t.} (a',t) \in \pi} t + \sum_{a' \text{s.t.} (a',t) \in \pi} |Threats(a')|,$$

$$ExecCost(a)^i = \sum_{a' \text{s.t.} (a',t) \in \pi} Cost(a') \cdot t,$$

where the relaxed plan π is computed by EvalAdd. The costs for $E(a)^r$ are defined in a similar way, and are computed by EvalDel. Figure 2 shows the main steps of EvalAdd and EvalDel, that we now describe.

The relaxed subplans used in EvalAdd and EvalDel are computed by RelaxedNumplan (see Figure 3). Given a set G of (propositional or numerical) goal facts and a numerical state $\langle I_l, N_l\rangle$, RelaxedNumplan computes an estimated minimal set ($Acts$) of actions required to reach G from $\langle I_l, N_l\rangle$. $SupportedFacts(l)$ denotes the set of propositional facts that are true after executing the actions in $\mathcal{A}$ at levels preceding l from the initial numerical state $\langle I_1, N_1\rangle$; $Pre(a)$ the preconditions of a; $Add(a)$ the propositional effects of a; V_{min} and V_{max} the sets of the min and max values of the numerical fluents, respectively. Such values are (monotonically) computed by ComputeMinMax from the values of N_l by applying the numerical effects of the actions in $Acts$.

After having computed the numerical state $\langle I_l, N_l\rangle$, in step 4 EvalAdd uses RelaxedNumplan to compute a relaxed subplan ($Rplan$) for achieving the preconditions of the new action a from $\langle I_l, N_l\rangle$. Steps 5 updates the propositional facts of I_l using $Add(a)$ and $Threats(a)$. In step 6, UpdateNumVal updates the values N_l of the numerical fluents using the numerical effects of a. Step 7 computes

[3] LPG pre-computes the action costs using the plan metric specified in the problem description (for more detail see [5]).

a relaxed plan for $Threats(a)$, possibly reusing the actions forming the previously computed relaxed subplan. Finally, step 8 adds the pair $(a, Times(a, g))$ to π, where $Times(a, g)$ is the minimum number of occurrences of a required to support the precondition g under consideration in the current search step.

EvalDel is simpler than EvalAdd, because the only new inconsistencies that can be generated by removing a from the current NA-graph are the precondition nodes supported by a that become unsupported. $UnsupFacts(a)$ denotes the set of these nodes. Step 4 computes a relaxed plan for $UnsupFacts(a)$ from the numerical state $\langle I_l, N_l \rangle$ computed by steps 2 and 3.

The set of pairs *(action, number of occurrences)* returned by RelaxedNumplan is derived by computing a relaxed plan ($Rplan$) for G, starting from a possibly non-empty input set of pairs (A) that can be reused to achieve the action preconditions or goals of the relaxed problem. RelaxedNumplan constructs $Rplan$ through a backward process where the pair chosen to achieve a (sub)goal g is a pair $(b, Times(b, g))$ obtained by combining the following requirements: (1) b is an helpful action for g; (2) all preconditions of b are reachable from $\langle I_l, N_l \rangle$; (3) reachability of the preconditions of b requires a minimum number of actions, estimated as the maximum of the heuristic number of actions required to support each precondition p of b at level l ($Num\_acts(p, l)$); (4) b subverts the minimum number of supported precondition nodes in $\mathcal{A}$ ($Threats(b)$); (5) b is applied the minimum number of times required to satisfy g ($Times(b, g)$). More precisely, ChooseAction(g) returns a pair $(b, Times(b, g))$ such that b is an action satisfying

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

where F is the set of the (positive) effects of the actions currently in *Acts*; A_g is the set of the actions with effect g (if g is a propositional precondition) or with an effect that reduces the gap in the numerical comparison of g (if g is a numerical precondition). A "relaxed" check of such a reduction is done using the values of V_{min} and V_{max}. For instance, suppose that g is `x > y`, that according to V_{min} `y` $= 25$, and that according to V_{max} `x` $= 15$. An action with an effect assigning 10 to `x` does not belong to A_g, while an action assigning 20 to `x` belongs to A_g.

$Num\_acts(p, l)$ is computed by *reachability analysis* using a polynomial algorithms similar to the one proposed in [5]. The main differences, that for lack of space here we do not describe in detail, concern numerical preconditions. In LPG-IPC3, the reachability analysis simply ignores the numerical preconditions of the actions examined, while the new version of the planner treats them as well. Moreover, in LPG-IPC3, the Num_acts-value of every unsupported numerical precondition is always 1, while in the new version it is estimated more accurately.

3.3 Search Restarts for Improving Plan Quality

LPG produces a succession of valid plans obtained by appropriately restarting the search when it finds a plan. Each plan is an improvement of the previous ones in terms of its quality. The first plan generated is used to initialize a new search for a second plan of better quality, and so on. This is a process that incrementally improves the quality of the plans, and that can be stopped at any time to give the best plan computed so far. Each time we start a new search, some inconsistencies are forced in the NA-graph representing the best plan Π computed so far, and the resulting NA-graph is used to initialize a new search.

For plan metric expressions requiring *minimizing* a numerical expression, LPG *removes* some actions from Π, preferring those with high execution costs. For plan metric expressions requiring *maximizing* a numerical expression, LPG *adds* some expensive actions to Π. The "HardNumeric" variant of `Satellite` used in the 3rd IPC is an example of domain requiring maximizing a numerical expression.

Domain	Problems solved		LPG-speed		LPG-quality	
	LPG	FF	better than FF	worse than FF	better than FF	worse than FF
Numeric						
Depots	21	19	13	8	15	1
DriverLog	20	16	9	11	16	0
Rovers	20	9	11	7	0	0
Satellite	19	13	9	10	13	0
Settlers	11	4	9	0	0	2
ZenoTravel	20	20	1	18	15	4
HardNumeric						
DriverLog	20	16	10	9	16	0
Satellite	20	14	6	14	14	0
`Total`	93.2%	68.5%	42%	47.5%	83.2%	5.9%

Table 1. Summary of the comparison between LPG and Metric-FF in terms of number of problems solved, CPU-time and plan quality.

In LPG-IPC3, both the two terms of the evaluation function E are positive quantities, and the planner is not capable of finding plans of good quality for maximization problems. In the new version of LPG, the sign of the term for the execution cost can be either positive or negative. Specifically, it is positive when the planner tries to minimize the plan metric, and it is negative when it tries to maximize it. This sign is automatically set, and allows LPG to find good quality plans for both minimization and maximization problems. In particular, as we will show in the next section, by appropriately setting the sign of the execution cost and using the simple restart strategy introduced above, LPG solves all problems of the HardNumeric `Satellite` domain very efficiently in terms of plan quality.

4 Experimental Results

The techniques presented in the previous sections are implemented in a new version of LPG. In this section, we give some experimental results illustrating the performance of LPG using the `Numeric` and `HardNumeric` variants of the 3rd IPC test problems. These problems belong to the domains `Depots`, `DriverLog`, `Rovers`, `Satellite`, `Settlers` and `Zenotravel`.[4]

The results for LPG correspond to median values over five runs for each problem considered. The performance of LPG was tested in terms of both CPU-time required to find a solution (LPG-speed) and quality of the best plan computed (LPG-quality), using at most 5 CPU-minutes.[5]

LPG-IPC3 does not include the extension of the neighborhood evaluation for handling numerical expressions that we have presented. In particular, in that version of the planner, the heuristic evaluation of each NA-graph in the neighborhood simply ignores the numerical preconditions that *become* unsatisfied as a negative side effect of removing the inconsistency under consideration. (E.g., if we add an action a at a level l of the current NA-graph $\mathcal{A}$, and a has the numerical effect `decrease(x,50)`, it could be that the numerical precondition `x > 60` of another action at a level after l becomes unsupported.) Moreover, LPG-IPC3 does not consider numerical preconditions when choosing the actions forming the relaxed plans.

The new heuristic functions EvalAdd, EvalDel, and RelaxedNumplan introduced in this paper lead to significant improvements of the performance of LPG. The percentage of the numerical problems solved by LPG-IPC3 is 59.9%, while this percentage with the new version of LPG is 93.2%; the percentage of the problems where the new

[4] For a description and formalization of these domains, see `www.dur.ac.uk/d.p.long/competition.html`.

[5] All tests were conducted on a PIII Intel 866 Mhz with 512 Mbytes of RAM.

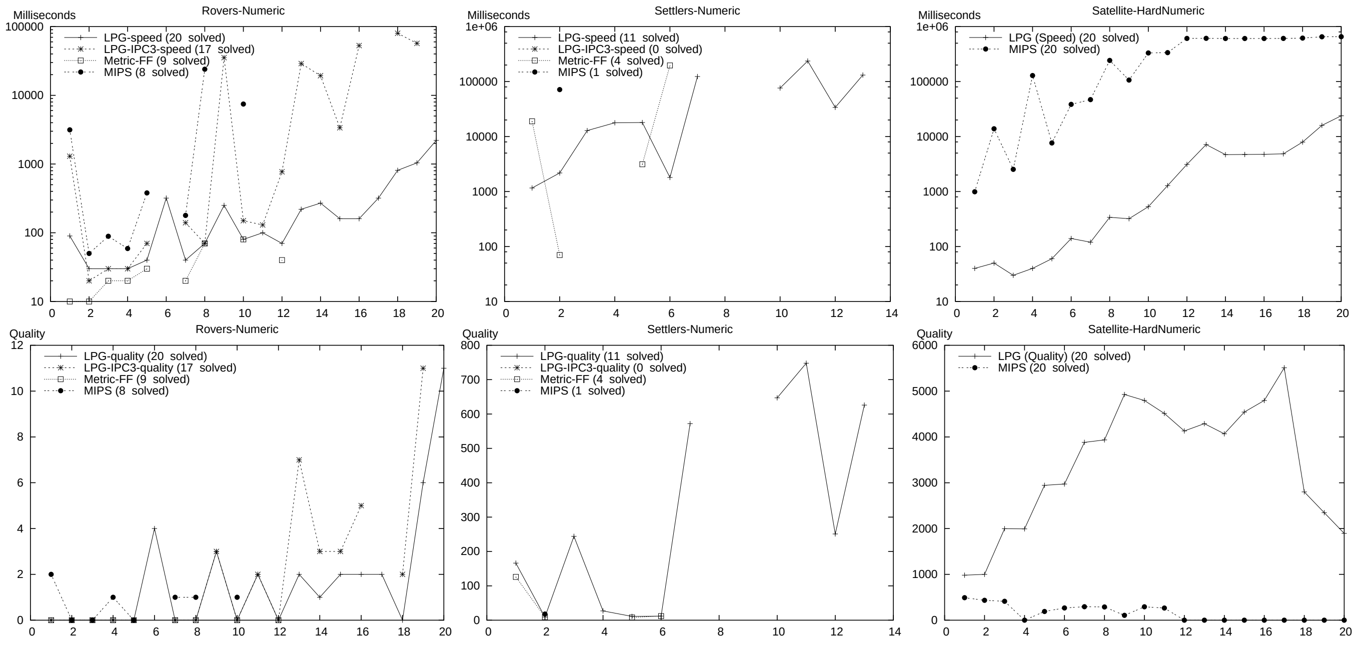

Figure 4. Number of problems solved and performance of LPG-speed (upper plots) and LPG-quality (bottom plots) compared with Metric-FF, MIPS and LPG-IPC3 in Rovers Numeric, Settlers Numeric, and Satellite HardNumeric. On the x-axis, we have problem names indicated with numbers. On the y-axis, we have CPU-time (log scale) or plan quality measured using the metric specified in problem formalizations. The plan metrics for Rovers and Settlers require to minimize a numerical expression, while the plan metric of the Satellite problems require to maximize a numerical expressions.

version of LPG is at least one order of magnitude faster than LPG-IPC3 is about 40%, while this percentage for LPG-IPC3 is 0%; finally, in terms of plan quality, the new version of LPG performs generally better than LPG-IPC3.

Table 1 compares the performance of LPG and Metric-FF [6]. In terms of CPU-time, LPG and Metric-FF perform similarly. The percentage of the problems where our planner is faster is 42%, while for Metric-FF this percentage is 47.5%. However, the percentage of the problems where LPG is at least one order of magnitude faster is 27.8%, while for Metric-FF this percentage is only 6.2%. Moreover, the percentage of the problems solved is 93.2% for LPG, 68.5% for Metric-FF. In terms of plan quality, LPG performs generally better than Metric-FF: the percentage of the problems where LPG produces a plan of better quality is 83.2%, while for Metric-FF this percentage is only 5.9%.

Figure 4 shows the performance of the new version of LPG compared with Metric-FF, MIPS[2] and LPG-IPC3 in three domains: Numeric Rovers, Numeric Settlers, and HardNumeric Satellite. The Rovers and Settlers problems require to minimize a numerical expression, while the Satellite problems require to maximize a numerical expression. LPG solves all 20 Rovers problems, Metric-FF 9, MIPS 8, LPG-IPC3 17 (but it is generally slower than the new version of LPG – up to two orders of magnitude). The quality of the plans found by LPG for the Rovers problems is better than or equal to the quality of the plans generated by the other planners. In Settlers, LPG solves 11 of the 20 test problems, Metric-FF 4, MIPS 1, and LPG-IPC3 0.

Regarding the Satellite problems, it should be noted that they admit, as special solutions, plans of quality zero. However, such plans are useless and not interesting, given the maximization goal of the problems. Since Metric-FF solves all these problems producing plans with quality zero, we have not considered it for this domain. MIPS was the only planner of the 3rd IPC capable of finding some plans with quality higher than zero for Satellite HardNumeric. However, in general, the quality of the plans found by the new version of LPG is much better than the quality of the plans generated by MIPS.

5 Conclusions

The capability of planning in domains involving numerical expressions is very important for addressing real-world problems. In this paper, we have presented the graph-based representation used by LPG for handling plans involving numerical quantities, and some new local search techniques for planning using this representation.

Experimental results obtained using the test problems of the 3rd planning competition show that our system performs very well with respect to Metric-FF and MIPS, and it significantly improves the version of LPG that took part in the competition.

ACKNOWLEDGEMENTS

Alessandro Saetti thanks the European Coordinating Committee for Artificial Intelligence for a travel grant.

REFERENCES

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

[2] Do, M., and Kambhampati, S. Sapa: A scalable multi-objective heuristic metric temporal planner. JAIR 20:155–194. 2003.

[3] Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. JAIR 20:61–124.

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

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

[6] Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *JAIR* 20:291–341.

[7] Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *JAIR* 20:1–59.

[8] McDermott D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–56.

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