Extending FF to Numerical State Variables

Jörg Hoffmann[1]

Abstract. The FF system obtains a heuristic estimate for each state during a forward search by solving a relaxed version of the planning task, where the relaxation is to assume that all delete lists are empty. We show how this relaxation, and FF's heuristic function, can naturally be extended to planning tasks with constraints and effects on numerical state variables. First results show that the implementation, Metric-FF, is competitive with other approaches to numerical planning, performing well against one of the most recent approaches on a numerical version of the *Logistics* domain.

1 Introduction

Four out of five awarded fully automatic planners in the AIPS-2000 planning systems competition were (at least partially) based on the idea of heuristic search [1]. In particular, the heuristic planner FF [8] performed best. While in the competition the domains were purely logical, i.e., STRIPS and ADL formulations, it is clearly important for real world applications to deal with more expressive constructs, like temporal actions with numerical resource requirements. As the language to be used in the AIPS-2002 competition, Maria Fox and Derek Long have therefore developed PDDL2.1 [5], which amongst other things incorporates the possibility to define numerical constraints and effects on a finite set of numerical state variables.

Some approaches have been made to do planning in more expressive formalisms (e.g. [6, 9, 10, 7]). One of the most recent ones, and a relative of FF, is the Sapa system, developed by Do and Kambhampati [4]. In FF, a heuristic estimate for a state s during forward search is obtained by extracting, in Graphplan [2] style, a *relaxed* plan that achieves the goals from s, where the relaxation is to assume that all delete lists are empty (this relaxation has first been proposed by Bonet et al. [3]). The length of the relaxed plan is taken as the (inadmissible) heuristic estimate [8]. In Sapa, which handles temporal actions as well as numerical variables, a temporal relaxed plan based on the same relaxation is extracted by using a temporal Graphplan style algorithm. Relaxed planning *completely ignores the numerical part* of the task, and heuristics are afterwards applied to estimate the relaxed plan's resource consumption [4].

In this work, we explain how the idea of ignoring delete lists as a relaxation can naturally be extended to tasks with numerical state variables. Briefly, the idea is to transform the task such that all numerical constraints are monotonic in a certain sense, and then to ignore, in the relaxation, all numerical effects that decrease the value of a variable. One can then solve the relaxed task, *combining the logical and numerical aspects*, by a natural extension of FF's heuristic function, preserving that function's theoretical properties. The algorithms are implemented in the Metric-FF planning system, using the heuristic function in a standard forward state space search. We present some preliminary evaluation of the system on the numerical *Logistics* variant introduced for the Sapa planner. Compared with Sapa, Metric-FF demonstrates superior runtime performance while finding plans of similar length.

The Metric-FF system is implemented to deal with arbitrary combinations of ADL and the numerical framework of PDDL2.1. For readability, we consider STRIPS tasks only. Section 2 presents the algorithms used in (original) FF's heuristic function. Section 3 gives notations for the numerical framework of PDDL2.1. In Section 4 we introduce our ideas and algorithms on a restricted form of numerical constraints and effects, while Section 5 explains how these techniques can be extended to more complex language constructs. Section 6 presents the experimental results, and Section 7 concludes and gives some outlook.

2 FF

We assume that the reader is familiar with STRIPS. By $pre(a)$ we denote an action's *precondition*, by $\mathit{eff}(a)^+$ the *add list* and by $\mathit{eff}(a)^-$ the *delete list*. A *task* is a triple (A, I, G) of action set, initial state, and goal set. A sequence of actions is a *plan for* (A, I, G) if, executed in I, it yields a state that satisifes all goals.

FF performs search in the space of all states that are reachable from the initial state. For any search state s, the *relaxed task to* s is solved. This is defined as (A^+, s, G), where A^+ is the original action set except that all delete lists are assumed to be empty. Solving (A^+, s, G) is done in Graphplan [2] style, i.e., by first building a relaxed planning graph and then extracting a plan from it. For ease of understanding the extended algorithms that we present later, we show the pseudo-code in Figures 1 and 2.

$P_0 := s$
$t := 0$
while $G \not\subseteq P_t$ **do**
  $A_t := \{a \in A \mid pre(a) \subseteq P_t\}$
  $P_{t+1} := P_t \cup \bigcup_{a \in A_t} \mathit{eff}(a)^+$
  if $P_{t+1} = P_t$ **then**
    fail
  endif
  $t := t + 1$
endwhile
$finallayer := t$

Figure 1. Building a relaxed planning graph for a state s.

The relaxed planning graph is built as a sequence of proposition- and action-layers. Proposition layer 0 is the current state, action layer 0 are all applicable actions, whose add lists make up proposition layer 1, and so on until either the goal or a fixpoint is reached. In the latter case, the heuristic value to s is set to ∞. In the former case, plan extraction is invoked. This is a simple for-next loop from the top to the bottom layer, making use of a sequence of goal sets G_t. The *level* of each proposition or action is the first layer where it appears. All goals are put into the goal sets at their respective layers. The plan

[1] Institut für Informatik, Georges-Köhler-Allee Geb. 52, 79110 Freiburg, Germany, e-mail: hoffmann@informatik.uni-freiburg.de

```
for t := 1, ..., finallayer do
    G_t := {g ∈ G | level(g) = t}
endfor
for t := finallayer, ..., 1 do
    for all g ∈ G_t do
        select a, level(a) = t − 1, g ∈ eff(a)^+
        for all p ∈ pre(a) do
            G_level(p) ∪ = {p}
        endfor
    endfor
endfor
```

Figure 2. Extracting a relaxed plan.

is extracted by selecting achieving actions at each layer, and inserting their preconditions into the respective goal sets (which are below the layer currently worked on, due to the way the graph is built).

The processes have the following theoretical properties, inherited from Graphplan [2]: if the graph encounters a fixpoint without reaching the goal, then (A^+, s, G) is unsolvable; the actions selected by plan extraction form a plan for (A^+, s, G). FF's heuristic value to s is set to the number of selected actions, i.e., to the *length of the relaxed plan*. This heuristic is used in a variation of hill-climbing [8].

3 Numerical State Variables

As said, we consider an extension of STRIPS with numerical constraints and effects. We briefly introduce notations for the PDDL2.1 framework defined by Fox and Long [5]. All sets are assumed to be finite if not said otherwise.

All constructs are based on a set P of propositions and a set V of variables. For notational reasons, we say $V = \{v^1, \dots, v^n\}$. A *state* s is a pair $s = (p(s), v(s))$ where $p(s) \subseteq P$ is a set of propositions and $v(s) = (v^1(s), \dots, v^n(s)) \in \mathbb{Q}^n$ is a vector of rational numbers (the obvious semantics being that $p(s)$ are the true propositions, and $v^i(s)$ is the value of v^i).[2]

An *expression* is an arithmetical expression over V and the rational numbers, using the operators $+$, $-$, $*$, and $/$. A *numerical constraint* is a triple $(exp, comp, exp')$ where *exp* and *exp'* are expressions, and $comp \in \{<, \leq, =, \geq, >\}$ is a *comparator*. A *numerical effect* is a triple (v^i, ass, exp) where $v^i \in V$ is a variable, $ass \in \{:=, +=, -=, *=, /=\}$ is an *assingment operator*, and *exp* is an expression (the effect right hand side). A *condition* is a pair $(p(con), v(con))$ where $p(con) \subseteq P$ is a set of propositions and $v(con)$ is a set of numerical constraints. An *effect* is a triple $(p(\mathit{eff})^+, p(\mathit{eff})^-, v(\mathit{eff}))$ where $p(\mathit{eff})^+ \subseteq P$ and $p(\mathit{eff})^- \subseteq P$ are sets of propositions, and $v(\mathit{eff})$ is a set of numerical effects such that $i \neq j$ for all $(v^i, ass, exp), (v^j, ass', exp') \in v(\mathit{eff})$. An *action* a is a pair $(pre(a), \mathit{eff}(a))$ where $pre(a)$ is a condition and $\mathit{eff}(a)$ is an effect.

The *result* of executing an action a in a state s is $result(s, a) = \mathit{eff}(a)(s)$ if $s \models pre(a)$, undefined otherwise. $s \models pre(a)$ holds iff $p(pre(a)) \subseteq p(s)$ and all numerical constraints in $v(pre(a))$ are met in s; a constraint $(exp, comp, exp')$ is met in s if the value of *exp* in s stands in relation *comp* to the value of *exp'* in s. $\mathit{eff}(a)(s)$ is the state s' where $p(s') = p(s) \setminus p(\mathit{eff}(a))^- \cup p(\mathit{eff}(a))^+$ and $v(s')$ is the value vector that results from applying all numerical effects in $v(\mathit{eff}(a))$, leaving unaffected values unchanged; applying a numerical effect (v^i, ass, exp) means to update the value of v^i in s with the value of *exp* in s, using the asignment operator *ass*.

A *planning task* is a tuple (P, V, A, I, G) where P and V are the propositions respectively variables used, A is a set of actions, I is a state, and G is a condition. A sequence of actions $\langle a_1, \dots, a_n \rangle \in A^*$ is a *plan* if the result of iteratively applying the sequence to I yields a state s such that $s \models G$.

In our algorithmic framework, we make distinctions between different degrees of expressivity that we allow in numerical constraints and effects, i.e., between different *numerical languages*. A numerical language is a tuple $(\mathit{Cons}, \mathit{Eff\text{-}ass}, \mathit{Eff\text{-}rh})$ where *Cons* is a possibly infinite set of numerical constraints, *Eff-ass* is a set of assingment operators, and *Eff-rh* is a possibly infinite set of expressions. A task (P, V, A, I, G) belongs to a language if all used constraints, assingment operators, and effect right hand sides are members of the respective sets. [3]

4 A Restricted Language

We first introduce our heuristic function for a restricted numerical language. This simplifies the notation of the algorithms, and clarifies the intuition behind our approach. Our implementation of the heuristic function for more expressive languages is a straightforward extension of the simple algorithms described in this section. The language we consider is the following:

$$\begin{aligned} (\ & \{(v^i, comp, c) \mid v^i \text{ variable}, comp \in \{\geq, >\}, c \in \mathbb{Q}\}, \\ & \{+=, -=\}, \\ & \{c \mid c \in \mathbb{Q}, c > 0\} \ ) \end{aligned}$$

That is, we allow constraints only to check if a variable is greater (greater or equal) than a constant, and restrict effects to increase or decrease the value of a variable by a positive constant. We name tasks that belong to this language *restricted tasks*.

FF's heuristic function can be very naturally extended to the above numerical language. To understand why that is so, let us go back to STRIPS for a moment. In the STRIPS setting, we relax the planning task by ignoring the delete lists. In effect, applying relaxed actions increases the number of true propositions in the state monotonically. As it is always preferable to have more propositions true, the relaxed task is easier to solve than the original task.

The crucial notion here is that of *monotonicity*: the relaxed actions are monotonic in that they only increase the number of true propositions; this simplifies the task because the preconditions of all actions (as well as the goal) are monotonic in that they are negation-free, i.e., if they are fulfilled in a state s then they will be fulfilled in any superstate $s' \supseteq s$.

Now reconsider our restricted numerical language above. What we do is, we extend the concept of (propositional) monotonicity to a concept of numerical monotonicity, meaning increasing values of variables. As for the numerical constraints, they are already monotonic in our restricted language: we compare (the values of) variables to constants by $\geq$ or $>$ comparators, so a constraint that is fulfilled in a state s will also be fulfilled in any state s' where $v^i(s') \geq v^i(s)$ for all i. As for the numerical effects, they either increase or decrease the value of a variable by a positive constant, so we can make the actions monotonic by *ignoring the decrease effects*. [4]

In combination with the propositional relaxation, the relaxed task is thus derived from ignoring all delete- and decrease-effects. The algorithm that builds a relaxed planning graph is depicted in Figure 3.

[2] We ignore, for readability reasons, the possibility given in Fox and Long's original language that a variable can have an undefined value until it is assigned one. Our methododology can be easily extended—and is in fact implemented—to deal with this case.

[3] In the definition of linear tasks that we give later, the allowed expressions depend on the structure of the planning task; so for formal precision *Cons* and *Eff-rh* would need to be *functions* from planning tasks into sets of constraints respectively expressions. The above notation yields a more readable presentation, while its meaning should be intuitively clear.

[4] We remark that Do and Kambhampati [4] make a few notes that seem to go into the same direction, but don't explore this further.

$P_0 := s, \forall v^i : max_0^i := v^i(s)$
$t := 0$
while $p(G) \not\subseteq P_t$ or $(v^i, \geq [>], c) \in v(G), max_t^i \not\geq [\not>] c$ **do**
$A_t := \{a \in A \mid p(pre(a)) \subseteq P_t,$
$\forall (v^i, \geq [>], c) \in v(pre(a)) : max_t^i \geq [>] c\}$
$P_{t+1} := P_t \cup \bigcup_{a \in A_t} p(\mathit{eff}(a))^+$
$\forall v^i : max_{t+1}^i := max_t^i + \sum_{a \in A_t, (v^i, +=, c) \in v(\mathit{eff}(a))} c$
if $P_{t+1} = P_t$ and
$\forall v^i : max_{t+1}^i = max_t^i$ or $max_t^i > maxneed^i$ **then**
fail
endif
$t := t + 1$
endwhile
$finallayer := t$

Figure 3. Building a relaxed planning graph for a state s, in our restricted numerical language.

Our idea, like in FF, is to solve the relaxed task for each search state s, and take relaxed solution length as a heuristic estimate. The algorithm in Figure 3 decides relaxed solvability, and can be used to extract a relaxed plan. Let us explain the algorithm. It extends FF's relaxed planning graph for STRIPS to our restricted language. Reconsider Figure 1. The extension is that, in addition to the proposition- and action-layers, we propagate a value max_t^i for each variable i through the layers t of the graph, denoting the maximum value v^i can have after applying t time steps of relaxed actions. The max_0 values are just the values of the variables in s. The goal is reached at layer t when the propositional goal is contained in P_t, and all numerical constraints can be met according to the max_t values. Likewise, for an action to be applied in layer t, its propositional as well as numerical preconditions must be possible according to P_t and the max_t values. The next proposition layer is obtained as before by adding the union of the respective add lists; the next max values are obtained by adding the sum of the respective increasing effects (in Figure 3, note that the constant c in the sum denotes the right hand sides of the effects increasing v^i). The graph reaches a fixpoint when there are no new propositions, and the max values of all variables have either not improved, or are already more than is needed: we define $maxneed^i$ as the maximum over all constants c such that there is a constraint $v^i \geq [>] c$ in the task (the maximum over an empty set being $-\infty$).

Proposition 1 *Given a restricted planning task (P, V, A, I, G), and a state s. If the algorithm depicted in Figure 3 fails, then the relaxed task to s is unsolvable.*

With the graph being in a fixpoint, no layer will ever reach the goals; if there was a relaxed plan of length t then the goals would be reached in layer $t + 1$; finished by contradiction.

We will see below that the opposite direction of Proposition 1 also holds, i.e., the relaxed planning graph reaches the goals if and only if the relaxed task is solvable. As we want to use the relaxed planning graph for computing a heuristic estimate to each search state, it is of crucial importance to us how much time we need for the computation.

Proposition 2 *Given a restricted planning task (P, V, A, I, G), and a state s. The algorithm depicted in Figure 3 terminates in time polynomial in the size of the task and the number of time steps built.*

Trivially, building a single relaxed graph layer is polynomial; in fact, it involves only a single sweep over the action set, checking for precondition inclusion and the status of numerical preconditions. This can efficiently be implemented in a manner similar to the original FF implementation [8]. More tricky is the question how many layers need to be built. The number can be exponential: imagine the tasks where some variable must be greater than x, and can be incremented only by 1; relaxed plan length is linear in x, i.e., exponential in a non-unary encoding of x. For a unary encoding, the number of time steps is polynomial. Generally, the number is polynomial in the size of the plan (equal to the number of parallel time steps). [5]

If the relaxed planning graph could not reach the goals then we set our heuristic value to ∞, as is justified by Proposition 1. Otherwise, we extract a relaxed plan by the algorithm depicted in Figure 4.

for $t := 1, \ldots, finallayer$ **do**
$p(G_t) := \{g \in p(G) \mid level(g) = t\}$
$v(G_t) := \{(v^i, \geq [>], c) \in v(G) \mid level(v^i, \geq [>], c) = t\}$
endfor
for $t := finallayer, \ldots, 1$ **do**
for all $g \in p(G_t)$ **do**
select $a, level(a) = t - 1, g \in p(\mathit{eff}(a))^+$
for all $p \in p(pre(a)), (v^i, \geq [>], c) \in v(pre(a))$ **do**
$p(G_{level(p)}) \cup = \{p\}$
$v(G_{level(v^i, \geq [>], c)}) \cup = \{(v^i, \geq [>], c)\}$
endfor
endfor
for all $(v^i, \geq [>], c) \in v(G_t)$ **do**
while $max_{t-1}^i \not\geq [\not>] c$ **do**
select $a, level(a) = t - 1, (v^i, +=, c') \in v(\mathit{eff}(a))$
$c := c - c'$
/* introduce a's preconditions as above */
endwhile
$v(G_{t-1}) \cup = \{(v^i, \geq [>], c)\}$
endfor
endfor

Figure 4. Extracting a relaxed plan for our restricted language.

The similarity to the original FF algorithm in Figure 2 is closer than the size of Figure 4 suggests. The extension is merely that we now additionally introduce numerical goals and subgoals $(v^i, \geq [>], c)$ requiring that the value of variable v^i is greater or equal (greater) than constant c at some graph layer. The numerical goals must be introduced for the goal constraints and for the numerical preconditions of actions; they are achieved through the selection of appropriate actions with increase effects. In more detail: the *level* of a numerical goal $(v^i, \geq [>], c)$ is the smallest t such that $max_t^i \geq [>] c$ holds; the numerical goals and the numerical preconditions of selected actions are inserted into the respective numerical goal sets; [6] a numerical goal $(v^i, \geq [>], c)$ at a layer t is achieved by selecting actions directly below in the graph that increase v^i by c', and substracting c' from c, until the goal can be achieved one layer earlier. The latter loop terminates due to the way we build the relaxed planning graph: if $max_t^i \geq [>] c$ then $max_{t-1}^i \geq [>] c''$ where c'' is the result of substracting all increase effects c' in layer $t - 1$ from c.

Proposition 3 *Given a restricted planning task (P, V, A, I, G), and a state s for which the algorithm depicted in Figure 3 terminates positively. The actions selected by the algorithm depicted in Figure 4 form a plan for the relaxed task to s.*

By selecting actions that achieve the propositional and numerical goals at each layer t, we guarantee that, after executing (the relaxed version of) the actions selected at the layers below, all these goals

[5] One can decide relaxed solvability in polynomial time by doing a kind of look-ahead for the numerical variables each time a proposition layer is equal to the previous one. We did not yet explore that direction in depth as it seems difficult to actually compute a relaxed plan based on that information; also, it is unclear how relevant the possible exponentiality is in tasks that are not especially constructed to provoke it. Addressing the issue is a topic for future work.

[6] When inserting a numerical goal for v^i it is checked whether such a goal is already in the respective goal set; if so, the stronger of both is taken.

are fulfilled, no matter in which order the actions selected at a single layer are arranged. So arranging the selected actions at each layer in an arbitrary order yields a relaxed plan to the task. [7]

5 Linear Tasks

We so far have a heuristic function for tasks in the restricted numerical language. Changing perspective, what we really want is a heuristic function for tasks with *arbitrary* numerical constraints and effects. In this work, we have restricted ourselves to a language where the monotonicity property—on which our relaxation and algorithms for the restricted language are built—can be achieved. We consider what we call the language of *linear* tasks. Let us define what this is.

Given a planning task (P, V, A, I, G). A variable $v^i \in V$ is a *task constant* if v^i is not affected by the effect of any action in A. An expression is a task constant if all variables occuring in it are task constants. Note that a task constant can be replaced by a rational number. An expression is *linear* if: for all sub-expressions $(exp * exp')$, either *exp* or *exp'* is a task constant; and for all sub-expressions (exp/exp'), *exp'* is a task constant. The class of linear tasks is:

$$(\{(exp, comp, exp') \mid exp, exp' \text{ linear expression}, comp \text{ arbitrary}\},\\ \{:=, +=, -=\},\\ \{exp \mid exp \text{ linear expression}\})$$

That is, we restrict ourselves to linear expressions, and the assignment operators $:=$, $+=$, and $-=$ (we will say some words on more expressive languages in the outlook). Linear tasks can be brought into a normal form where all constraints, and effects whose right hand side depends on variables, are monotonic (in the sense explained below). The language is the following, which we refer to as the language of *linear normal form*, or *LNF* tasks:

$$(\{(\textstyle\sum_{j \in X} c^j * v^j, \geq [>], c) \mid c^j > 0\},\\ \{:=, +=\},\\ \{\textstyle\sum_{j \in X} c^j * v^j + c \mid c^j > 0\})$$

Here our constraints compare weighted sums of variables ($X \subseteq \{1, \dots, n\}$) to constants via $\geq$ or $>$ comparators, with all weights in the sum being positive (negative weights can be translated by introducing inverse variables, as described below). The only assingment operations are $:=$ and $+=$ ($:=$ will be treated as a special case in our algorithms; we get rid of $-=$ only for notational simplicity: this operator can be translated to $+=$ by multiplying the right hand side with (-1), see below). Effect right hand sides are positive weighted sums plus a constant (again, negative weights being translated). As all weights are positive, we have the following monotonicity properties. The constraints are monotonic (just like before) in that, if fulfilled in a state s, they will also be fulfilled in any state s' where $v^i(s') \geq v^i(s)$ for all i. The effect right hand sides are monotonic in that, given states s and s' as above, their value in s' is greater than, or equal to, their value in s.

For relaxation, we will make the actions monotonically increasing simply by ignoring those effects that decrease the value of the affected variable. More on this later. First, we show how a linear task can be transformed into an LNF task.

5.1 LNF Transformation

We start with an arbitrary linear task. Our first steps are the following: replace $(exp, =, exp')$ constraints with $(exp, \leq, exp')$ and $(exp, \geq, exp')$; replace $(exp, <, exp')$ with $(exp', >, exp)$ and $(exp, \leq, exp')$ with $(exp', \geq, exp)$; replace $(exp, \geq [>], exp')$ with $(exp - exp', \geq [>], 0)$; replace decreasing effects $(v^i, -=, exp)$ with $(v^i, +=, (-1) * exp)$. We end up with the following language:

$$(\{(exp, \geq [>], 0) \mid exp \text{ linear expression}\},\\ \{:=, +=\},\\ \{exp \mid exp \text{ linear expression}\})$$

In the next step, we normalize all linear expressions to weighted sums (by replacing task constants with rational numbers, then summing up the occurences of each variable). [8] We end up with:

$$(\{(\textstyle\sum_{j \in X} c^j * v^j, \geq [>], c) \mid c^j \neq 0\},\\ \{:=, +=\},\\ \{\textstyle\sum_{j \in X} c^j * v^j + c \mid c^j \neq 0\})$$

The only difference to LNF is now that the weights c^j can have negative values (redundant variables with weight 0 are ignored). To get rid of negative weights, we now perform the only non-trivial step of the transformation. We introduce, for each variable v^i that has a negative weight $c^i < 0$ in some sum, a new variable $-v^i$ that always takes on the inverse value of v^i. We can then replace v^i in the respective sum with $-v^i$, and multiply its weight with (-1). We set the initial value of $-v^i$ to $(-1) * v^i(I)$. For each effect $(v^i, := [+=], \sum_{j \in X} c^j * v^j + c)$ we introduce the new effect $(-v^i, := [+=], \sum_{j \in X} (-1) * c^j * v^j + (-1) * c)$. Note that the new effects can introduce new negative weights $(-1) * c^j$; however, if the negative value of the respective variable v^j is already translated into the inverse counterpart $-v^j$ then we can use the counterpart with the positive weight c^j; each varable needs to be translated at most once. Upon termination, the task is in LNF.

5.2 LNF Relaxed Planning Graphs

The algorithms for restricted tasks, which we have presented in detail in Section 4, can easily be extended to LNF tasks. We do not give full details here, but describe how we have implemented the extension.

First, we need to define what exactly our relaxation is. As said before, we want to ignore all effects that decrease the value of a variable. This can however depend on the state in which an action is executed (as an example, the effect $(v^i, +=, v^j)$ is increasing in states s where $v^j(s) > 0$ but decreasing where $v^j(s) < 0$). So we can not "statically" relax a task by ignoring parts of its specification. Instead, our relaxation is "dynamic": we relax the *state transition function* $result$ into the function $result^+$ that applies only the positive propositional effects, and only those numerical effects that increase the value of the respective variable (note that in STRIPS and our restricted language this comes down to exactly the relaxations we have used before). The relaxed task can be solved by a straightforward implementation of relaxed planning graphs.

Building a relaxed planning graph is done analogous to the algorithm in Figure 3. To deal with weighted sums, we introduce new artificial variables: one artificial variable for each constraint where at least two variables are compared to a constant ($|X| > 1$), and one artificial variable for each effect right hand side that depends on at least one variable ($|X| > 0$)—the other cases belong to the restricted langugage. The initial values of all artificial variables are set to the

[7] In fact, the algorithms can be viewed as a relaxed version of the Resource-IPP system [9].

[8] We also move the constant parts of constraint left hand sides back to the right hand sides—algorithmically, comparing to an arbitrary constant or to 0 does not make any difference.

values of the respective weighted sums in the state at hand. The graph for the non-artificial variables is built based on the $+=$ effects exactly like before, where the effect right hand side values at action layer t are determined by inserting the respective max_t values into the respective weighted sums, and effects with a negative right hand side are ignored. Each time a proposition layer $t+1$ is built, two additional steps are carried out. To take care of $:=$ effects, for each variable v^i we determine the maximum $:=$ effect right hand side (on v^i) that is yet in the graph; if that value is higher than max^i_{t+1} then max^i_{t+1} is replaced with the higher value. As for the artificial variables, once the max_{t+1} values of all non-artificial variables are determined, their max_{t+1} values are set to the value of the respective weighted sums, when inserting the max_{t+1} values for all variables. As before, the graph reaches its fixpoint when there are no propositional changes, and all (artificial and non-artificial) variables have either not changed, or have reached their $maxneed$ value.

Extending relaxed plan extraction, Figure 4, from a graph that has reached the goals can be done as follows. If a numerical goal $(\sum_{j \in X} c^j * v^j, \geq [>], c)$ for an artificial variable shall be inserted into layer t, then insert the goals $(v^j, \geq, max^j_t)$ for all $j \in X$. This way the weighted sum takes on the maximum possible value, which is guaranteed to be enough (in fact one could use heuristics here to make the goal requirements for v^j weaker). If a numerical goal shall be achieved at some layer, and there is a $:=$ effect below whose value is high enough, then select the respective action. If an effect is selected whose right hand side is an artificial variable v, then—in addition to the respective action's preconditions—introduce a numerical goal requiring that v takes on its max value.

Say we are given an LNF planning task (P, V, A, I, G), and a state s. If building the relaxed planning graph for s fails then the relaxed task to s is unsolvable—just like before, encountering the fixpoint implies that the goals won't be reached in any layer, but the existence of a relaxed plan of length t implies reaching the goals in layer $t+1$. The other way round, just like before the actions selected by plan extraction form a relaxed plan for s, in an arbitrary linearization—by supporting the numerical goals in the way we do, we guarantee that all variables have at least as high values as we require; finished by monotonicity of constraints and effect right hand sides, and by the fact that $result^+$ ignores all effects that decrease variable values.

6 Results

As said, we use relaxed plan length as a heuristic estimate in forward state space search. The search algorithm we use currently is a standard weighted A^*, where the weight (of the heuristic function) can be set as an option. We present some preliminary evaluation by running the planner against Sapa on a collection of numerical *Logistics* tasks where trucks and planes use (continuous amounts of) fuel and can be refueled at airports. The (temporal) domain and a suite of 9 tasks have been introduced with Sapa [4]. For Metric-FF, we have created a non-temporal version of the domain. The original suite is easily solved by both planners, which is why we have randomly generated tasks with increasing numbers of cities and packages. We ran both planners on a Linux workstation with 128 M Byte memory running at 850 MHz. Sapa uses a weighted A^* search with weight 5, so we have used the same weight in Metric-FF. See the data in Figure 5, showing runtime (in seconds), number of evaluated states, and plan length (i.e., number of steps) for both planners, against increasing number of cities (#c) and packages (#p). Metric-FF demonstrates clearly superior runtime performance (on the task with 5 cities and 6 packages, Sapa ran out of memory), while both systems find plans of roughly similar length. It should be noted, however, that Sapa is implemented in Java while Metric-FF is implemented in C. Also, Sapa finds concurrent plans while Metric-FF's plans are purely sequential. Looking at the number of evaluated states, Metric-FF is still superior, but far less than the runtime values suggest.

task		Metric-FF			Sapa		
#c	#p	time	states	steps	time	states	steps
2	2	0.01	66	4	0.48	124	4
2	3	0.02	178	12	1.48	307	12
2	4	0.03	397	28	3.06	515	27
3	3	0.03	185	10	1.78	275	10
3	4	0.03	232	16	4.16	487	18
3	5	0.12	1097	17	13.32	1461	17
4	4	0.09	474	21	15.50	1234	24
4	5	0.17	863	30	26.78	1675	32
4	6	0.19	1172	33	23.75	1596	30
5	5	0.17	827	23	27.16	1557	25
5	6	0.33	1388	43	-	-	-
5	7	0.66	2179	49	193.73	6702	60

Figure 5. Results on our random numerical *Logistics* suite.

7 Conclusion and Outlook

We have presented an extension of FF's heuristic function to a language including numerical constraints and effects over linear functions of numerical state variables. The approach is natural in that it generalizes the concepts of FF's original heuristic function, preserving the function's theoretical properties. Using the heuristic in a standard weighted A^* algorithm, the implementation Metric-FF is competitive with Sapa in a variant of the *Logistics* domain.

Based on the presented work, the author has—between submission deadline and final version deadline—extended FF's other heuristic techniques [8] to numerical state variables. The resulting system took part in the AIPS-2002 competition, achieving excellent results. See the competition web-page at `http://www.dur.ac.uk/d.p.long/competition.html` for more information.

Presumably, the presented heuristic techniques can easily be extended to durational constructs—the obvious approach would be to combine temporal Graphplan algorithms with the numerical Graphplan algorithms used here. Designing, implementing, and evaluating such a planning system is an open topic for future work.

REFERENCES

[1] Fahiem Bacchus, 'The AIPS'00 planning competition', *The AI Magazine*, **22**(3), 47–56, (2001).

[2] Avrim Blum and Merrick Furst, 'Fast planning through planning graph analysis', *Artificial Intelligence*, 90(1-2):279–298, 1997.

[3] Blai Bonet, Gábor Loerincs, and Héctor Geffner, 'A robust and fast action selection mechanism for planning', in *Proc. AAAI-97*, pp. 714–719. MIT Press, (July 1997).

[4] Minh. B. Do and Subbarao Kambhampati, 'Sapa: A domain-independent heuristic metric temporal planner', in *Proc. ECP-01*, pp. 109–120.

[5] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains, 2001. Unpublished Manuscript.

[6] M. Ghallab and H. Laruelle, 'Representation and control in IxTeT, a temporal planner', in *Proc. AIPS-94*, pp. 61–67, Chicago, IL, (1994). AAAI Press, Menlo Park.

[7] Patrick Haslum and Héctor Geffner, 'Heuristic planning with time and resources', in *Proc. ECP-01*, pp. 121–132.

[8] Jörg Hoffmann and Bernhard Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[9] Jana Koehler, 'Planning under resource constraints', in *Proc. ECAI-98*, pp. 489–493, Brighton, UK, (August 1998). Wiley.

[10] Ioannis Refanidis and Ioannis Vlahavas, 'Heuristic planning with resources', in *Proc. ECAI-00*, pp. 521–525, Berlin, Germany, (August 2000). Wiley.