NGP: Numerical Graph Planning

Joseph Zalaket and **Guy Camilleri**[1]

Abstract. Planning systems such as FF-Metric [3] and SAPA [2] are able to deal with numerical knowledge to solve resources constraints added to the symbolic planning domain. This paper presents the NGP (Numerical Graph Planning) that is able to solve totally numerical and/or symbolic planning domains. We propose a new action representation to support numerical conditions and effects, where we allow a non-restricted function application for numerical update. NGP guides its search using a heuristic derived uniformly for numerical and symbolic knowledge.

1 Introduction

Most of the real world problems involve numerical handling. Planning systems like FF-Metric [3] and SAPA [2] are able to handle numerical resources, but in most cases the resources are only used as auxiliary constraints added to the symbolic planning domain. However, real world problems require a more complicated numerical treatment. To manipulate a robot in a non flat territory or in the space there is need to support trigonometric functions. Similarly, functions are needed to plan the deployment of an army in different zones of the world. These types of problems need a totally numerical handling and they are normally solved by mathematical approaches. This paper presents the NGP planner and propose a method to handle discrete numerical quantities, as well as numerical goals in planning.

NGP extends the representational ability of the classical planning graph of GraphPlan [1] by introducing an approach to multi-valued the planning graph facts to keep trace of the update functions effects. This multi-valued approach allows keeping real value levels for numerical knowledge, which allows the calculation of a uniform heuristic for mixture of numeric and symbolic facts. NGP uses on-the-fly actions instantiation to deduce the appropriate numerical actions arguments during the search process.

In the second section we present the domain modeling extension. The actions instantiation and the numerical objects handling are described in the third section. In the fourth section we present the heuristic derivation for NGP and its main search mechanism, before ending with the conclusion.

2 Language

The language used in NGP is a slightly modified subset of PDDL2.1 language. The main extension to PDDL2.1 is the introduction of the update functions to handle the numerical knowledge. This extension allows the use of mathematical functions (like COS, SIN, SQRT,...) and user defined functions. The control flow (conditional statements and loops) can be used within an update function to hold up complex numerical computation.

[1] IRIT CCI-CSC, Université Paul Sabatier, 118 route de Narbonne, 31062 Toulouse Cedex 4, FRANCE. email: {zalaket, camiller}@irit.fr

2.1 Domain representation.

A planning domain is represented by a tuple D=$(X, C, F_p, R_p, F_u, R_c)$ where: X is the set of variables, C is the set of constant symbols, F_p is the set of functional symbols, R_p is the set of relational symbols, F_u is the set of update functions symbols and Rc=$\{=, \neq, <, \leqslant, >, \geqslant\}$ is the set of comparators.

- If $x \in X \Rightarrow x$ is a term. A term could be a domain object as block A,B,... or a constant value as 1,2,....
- If $c \in C \Rightarrow c$ is a term.
- If $f_p \in F_p$ with arity j, and $t_1, t_2, \ldots, t_j$ are terms $\Rightarrow f_p(t_1, t_2, ..., t_j)$ is a term. *We note by $\mathcal{N}$ the set of the numerical state variables of the domain,* for $t_1, t_2, \ldots, t_j \in C$.
- If $r \in R_p$ with arity i, and $t_1, t_2, \ldots, t_i$ are terms $\Rightarrow r(t_1, t_2, ..., t_i)$ is a literal. *We note by $\mathcal{P}$ the set of the literals of the domain*, for $t_1, t_2, \ldots, t_i \in C$.
- If $f_u \in F_u$ with arity n and $l_1, l_2, \ldots, l_n \in \mathcal{P} \cup \mathcal{N}$ are arguments of f_u, such that $l_i \in \mathcal{N}$ *and* $l_i \leftarrow f_u(l_1, l_2, \ldots, l_n) \Rightarrow f_u(l_1, l_2, \ldots, l_n)$ is an update function assigned to the numerical state variable l_i. *We note by $\mathcal{F}$ the set of the update functions of the domain.*

2.2 Problem definition

A planning problem is defined as a tuple P=(*I,O,G,*D) where: *I* is the initial state, *O* is the set of operators defined on the domain D and *G* is the set of goal satisfaction conditions.

i. $I \in S$ (the state space) such that: S=$\{(\alpha, \beta)/\alpha \in \mathcal{P},\ \beta : \mathcal{N} \rightarrow \mathbb{R}\}$

ii. An operator o $\in$ *O* is represented by the 5-tuple (CON, PRE, ADD, DEL, UPD) where:

CON is the constraints list. The constraints are tested before the action instantiation to avoid instantiation for inconsistent arguments. A constraint $cn \in CON$ is a triple $cn = (t, r, t')$ where: the term $t \in X$, $r \in R_c$ and $t' \in X \cup C$.

PRE = $\text{PRE}_\text{P} \cup \text{PRE}_\text{N}$ is the list of preconditions.

$\text{PRE}_\text{P} \subseteq \mathcal{P}$ are propositional preconditions.

PRE_N are numerical (functional) preconditions, a precondition $p_N \in \text{PRE}_\text{N}$ is a triple $p_N = (v, r, s)$ where $v \in \mathcal{N}$, $r \in R_c$ and s is a term ($s \in X \cup C \cup \mathcal{N}$).

ADD, DEL $\subseteq \mathcal{P}$ are respectively the list of propositional additions and the list of propositional deletions.

UPD is the list of update, made up of numerical state variables assigned to update functions. $u \in$ UPD is a triple $u = (v, ass, f)$ where $v \in \mathcal{N}$, $ass = \{:=\}$ is the assignment and $f \in \mathcal{F}$.

iii. *G* is the set of conditions satisfying the goal made up by propositional and numerical conditions (like the PRE list in (ii)).

3 Action Model

In a numerical planning problem many world objects are not explicitly defined but could be retrieved from the domain.
Definition-1: A variable object $v_o \in \mathcal{N}$ is a function that implicitly represents zero or one or several world objects $o_i \in C$. Each time an update is applied to the variable v_o, this variable takes a new value, and by consequence a new implicit world object is added to (or deleted from) the state space.
Transforming variable objects in a domain into explicit world objects consists of finding out the objects added (or deleted) progressively from the state space each time a numerical update takes place. In a propositional state space, the actions should be instantiated from the outset as compared to all the world objects in a total order planning process. By contrast in a numerical state space, actions instantiation could be increasingly accomplished each time a numerical effect "UPD set" is applied.
Definition-2: A numerical state variable $v \in \mathcal{N}$ assigned to an update function $f \in \mathcal{F}$ is an implicit parameter of the action having $v := f$ in its UPD list.
Lemma-1: A variable object $v_o \in \mathcal{N}$ is an implicit action parameter of (a) certain action(s).
A variable object is assigned to an update function of one (or several) action(s), and thus it is an implicit action parameter of this (or these) action(s) according to definition-2.
Definition-3: Any numerical state variable $l \in \mathcal{N}$ belonging to the arguments of an update function $f \in \mathcal{F}$ is an implicit action parameter of the action having $v := f$ in its UPD list, such that $v \in \mathcal{N}$.
This approach compared to the symbolic representation has the advantage of instantiating only what is needed as actions for problem resolving. Some partial order planning systems use the least commitment to avoid instantiating actions from the outset. In NGP actions are instantiated progressively during the planning process.
As the use of functions and numerical handling makes the planning process undecidable, we have added a lower limit and an upper limit for each numerical type. For non-monotone functions we have added a step variation ϵ to approximately match the calculated number with the upper or lower bounds limits.

4 NGP planner

The main search algorithm of the NGP planner is a variation of hill-climbing algorithm guided by a heuristic *h* derived from a relaxed planning graph. If the hill-climbing fails to find a solution, then an A* search algorithm takes place.
The NGP heuristic is calculated for each new state in the search space and it is based on the planning graph of GraphPlan-Style [1]. The use of the planning graph gives the possibility of non-restricting the update functions included in numerical effects (UPD list), as update functions are only executed in forward pass, also conditional update could be included in update functions. Consequently, the update functions could be one-way functions (non inversible or non bijective). Applying the Update list of an action leads to add the original version of the concerned facts (original values before update) to the Del list, then to add the updated version of these facts (new values after update) to the Add list. With this graph structure we can manage the propositional facts and the numerical ones in an identical way instead of calculating two different heuristics as is the case of the most of other planners treating numerical knowledge [3], [2]. In NGP we use the STAN [5] datastructure for an explicit graph construction.
The construction of the relaxed graph is done as follows:
Given a state S, the graph is built from S by applying the relaxed actions whose preconditions hold in S. An action is relaxed by ignoring its delete list, and by applying its update list as if it adds new state variables values. The actions application leads to a second fact layer in the graph, which contains all the facts added to the graph by applying the ADD list and the UPD list of the actions. As in GraphPlan we add "noop" actions to include the facts of fact-layer i-1 into the fact-layer i. This transforms the numerical state variables in the graph into multi-valued state variables to include values of previous layers. Again relaxed applicable actions are applied and so on, until we reach a fact-layer that contains all goals or until a fix point in the graph is reached. In the latter case the returned heuristic is infinite as we consider that there is no solution. In the former case a relaxed plan extraction process should take place to calculate the heuristic value. The resulting relaxed graph consists of a bi-layered graph, fact-layers and action-layers. A fact-layer consists of two types of facts: the propositional facts and the numerical facts. The numerical facts are multi-valued in the graph, in a way that every time an update aims to change the value of a numerical fact (state variable), this change is added as a new value to the fact.
Once the graph is constructed up to the goals, we can extract the relaxed plan. Each goal in the final fact layer (the layer that contains all goals), is replaced by the preconditions of the best action that adds it from the previews layer, in its turn the action is added to the list of relaxed plan. These preconditions become the goals of the current layer, and the process continues backward until reaching the initial layer. The length of the resulting extracted relaxed plan constitutes the heuristic *h*. $h = \sum_{i=0}^{finallayer-1} |Oi|$ where [O_0 ,..,$O_{finallayer-1}$] is the relaxed plan ([4]).

5 Conclusion

We have presented an ongoing work of a domain independent planning system able to solve numerical and symbolic domains or a combination of both. We have proposed a new action representation where the numerical effects are separated from the propositional ones and they can be developed by non-restricted update functions. We have also added a constraint list to the action definition, which serves to avoid useless actions instantiations. Our main objective in the presented work was to allow the definition of domains closer to the real world, where objects are not obligatory symbolic as for STRIPS, but they also can be retrieved from numerical functions. NGP is implemented in Java. The current implementation is used to test the capacity of NGP to deal with numerical domains. Some tests have been done on problems like the ferry (numerical version), the Water Jug , the manufacturing and the army deployment domains.

REFERENCES

[1] Avrim L. Blum and Merrick L. Furst, 'Fast planning through planning graph analysis', *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI95)*, 1636–1642, (1995).
[2] Minh B. Do and Subbarao Kambhampati, 'Sapa: A domain-independent heuristic metric temporel planner', *In Proceedings of the European Conference on Planning*, (2001).
[3] J. Hoffmann, 'Extending FF to numerical state variables', *In Proceedings of the 15th European Conference on Artificial Intelligence, Lyon, France*, (2002).
[4] J. Hoffmann and B. Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Artificial Intelligence Research*, **14**, 253–302, (2001).
[5] D. Long and M. Fox, 'Efficient implementation of the plan graph in STAN', *Journal of Artificial Intelligence Research*, **10**, 87–115, (1999).