# MORE EXPRESSIVE PLANNING GRAPH EXTENSION

Joseph Zalaket and Guy Camilleri
*Département informatique, Université Saint-Esprit de Kaslik,B.P. 446 Jounieh, Liban*
*IRIT CSC, Université Paul Sabatier,118 route de Narbonne, 31062 Toulouse, France*

Keywords: AI Planning, planning graph, numerical planning, functional planning.

Abstract: The most of the real world problems have characteristics that can not be handled like pure propositional expressions. Many classical planners have been extended to deal with more expressive problems which require features like temporal actions, numeric functions, conditional effects, ... etc. Extensions have been made to planners like (D. and D., 1999), MetricFF02, (Do and Kambhampati, 2001), which use the planning graph as their base structure. However, each of these extensions deals with the one or the other of the features required for more expressive planning problems without being able to integrate all or at least an important combination of these features. In this paper, we propose an extension to the planning graph structure to deal with more expressive planning problems that can contain real word data characteristics, such as: symbolic, temporal and numeric data.

## 1 INTRODUCTION

The domain independent-planning still a complex problem. The most of the domain-independent planners are able to solve symbolic planning problems but, real-world problems necessity dealing with more complex types of data such as temporal and numeric data. This kind of data requires features like numeric function and temporal actions to be added and supported by the planning process. Some planners have been extended to deal with some real world knowledge, and thus, to support one or the other of these features. Planners like SAPA (Do and Kambhampati, 2001) and Metric-FF (Hoffmann, 2002) have extended their heuristic search to deal with numeric knowledge. However, to combine multiple features which can be necessary to solve some more real-world problems in a domain-independent planner still a complex task. In the other hand, many planners are using the planning graph structure as their base for exploring the state space due to its compact structure. After its introduction in Graphlan (Blum and Furst, 1995) as its basic expansion graph, the planning graph became the subject of multiple extensions to solve directly some more expressive problems like temporal Graphplan (TGP) (Smith and Weld, 1999)or to serve as a base to calculate heuristic function for such problems like in like SAPA (Do and Kambhampati, 2001) and Metric-FF (Hoffmann, 2002). In this paper we propose a general extension to the planning graph, which allow it to deal with functions application. The functions that we propose can be incorporated into the action effects and/or the action preconditions. This extension allows the planning graph to support all types of features that are necessary to deal with more expressive planning problems, such as problems that contain numeric and temporal knowledge. In our extension we introduce the application of functions during the planning graph expansion, in a way that the results of these functions can be incorporated within the nodes of this graph and thus they can be treated as any other type of nodes. Using functions application will allow us to deal with any combination of features required for solving some real world problems as the functions that we propose can be written in any programming language without restrictions to their implementation. Therefore, the extended planning graph can be used directly as an expansion phase graph as in Graphlan-like planners (Blum and Furst, 1995), (Smith and Weld, 1999) to

Zalaket J. and Camilleri G. (2007).
MORE EXPRESSIVE PLANNING GRAPH EXTENSION.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 424-427

solve planning problems or it can also be used as the base graph to calculate heuristics as in FF-like planners (Hoffman, 2001), (Hoffmann, 2002).

We first present the motivation for our work in section 2. We then present an overview of the planning graph structure in section 3. We present the extension to the definition of the planning problem in section 4. We explain the planning graph extension to handle numerical variables and functions application in section 5. We finally conclude this paper in section 6.

## 2 MOTIVATION

Many extensions have tried to adapt the planning graph to solve more expressive planning problems than those that can be expressed in pure propositional STRIPS language. Real-world problems have complex characteristics. Most of these characteristics can be modeled with the PDDL3.0 (A. and D., 2005) language which is the mostly used by recent planners. Therefore, PDDL3.0 gives the possibility of expressing optimization criterion such that maximization or minimization of resource consumption, without being able to represent all king of resource handling. The extension that we propose to the planning graph structure aims to allow this graph to supprot the execution of functions. Functional Strips (Geffner, 1999) and FSTRIPS (Schmid et al., 2000) have studied the integration of functions to solve planning problems. Functional Strips and FSTRIPS use functions instead of relations, which allow reducing the generated literals in a planning problem. In our work we introduce the function application which allow us to represent complex numerical and conditional effects, as well as complex preconditions and goals. For example, to plan the motion of a robot that is expected to travel from an initial position to a target position in the presence of physical obstacles in its environment. To avoid collisions between the robot and the obstacles we have to use trigonometric functions that allow the robot to follow a circular path in the environment (BAK M. and O., 2000). Trigonometric expressions are simple to be solved by classical programming languages, but they can not be expressed in classical AI planning languages like PDDL3.0. For this reason our extension to th planning graph is based in the integration of functions written in any programming language. Inspiring from the representation of functions in object-oriented databases. We add to the problem definition written in PDDL3.0 the declaration of external functions by specifying their execution path. In addition to their capabilities of solving numerical problems like the circular movement of a robot, external function are able to handle all kind of conditional and probabilistic effects. To avoid all kind of side effects we restrict all the formal parameters of a function to be constant, which mean a function can only return a value without updating any state variable. This returned value is affected to one state variable at a time.
The use of non-bijective functions requires that their application in the search space could be done only in forward. As the planning graph is constructed in progression, thus it can support the application of this kind of functions.

## 3 PLANNING GRAPH

Because The planning graph introduced in Graphplan (Blum and Furst, 1995) is the basis of our extension, we briefly summarize the planning graph structure without detailing how it is used by the Graphplan algorithm.

A planning graph consists of a sequence of levels, where level 0 corresponds to the initial state. Each level contains a set of literals and a set of actions. The actions of a level are those that have their preconditions satisfied and mutually consistent by the literals of the same level. The literals of a next level i+1 are the effects of the applicable actions of previous level i. The literals that satisfy the preconditions of action instances at the same level are connected with these actions via direct edges. Also, the action instances are connected via direct edges with their literals effects at the next level.

The planning graph is expanded up to a level in which all the literals of the goal are present and consistent. Then, a solution extraction phase can start in a way that for each sub-goal at the last level n, it tries to find an action instance at level n-1 that has this sub-goal as an add effect. The preconditions of the selected action instances become the new set of sub-goals at level n-1 and so on until reaching the initial state (level 0).

## 4 PLANNING PROBLEM EXTENSION

Each distinct instance of the world is called a state, denoted by $s$. The set of all possible states is called a state space $S$. $S$ is formed by two disjoint sets: a set of logical propositions $P$ and the set of numeric variables $N$, such that $S = \{(p,n)/p \in P, n : N \rightarrow \mathbb{R}\}$.

We denote by $P(s)$ the subset of logical propositions of a state $s$ and by $N(s)$ the subset of its numeric variables.
A state transition functions $t$ transforms $s$ into $s' = t(s,a)$ by 3 ways:

- Adds logical propositions to $P(s)$
- Removes logical propositions from $P(s)$ (when propositions become false)
- Assigns new values to existing numeric variables in $N(s)$

An action $a$ is defined as a tuple $\langle arg, pre, eff \rangle$, where:

- $arg$ is the list of arguments made by variables which represent constant symbols and/or numeric variables in $N$.
- $pre = pre_P \cup pre_N$ is the list of preconditions.

  $pre_P$ defined over $P$ are propositional preconditions

  $pre_N$ are numeric preconditions, such that: $\forall c \in pre_N$, $c = (n \theta g)$, where $n \in N$, $\theta \in \{<, \leq, =, >, \geq\}$ and $g$ is an external function or an expression.
  ***Definition-1:*** *An expression is an arithmetic expression over N and the rational numbers, using the operators +, -, * and /.*
  ***Definition-2:*** *An external function is written in a high level programming language on the form of type $f(n_1, n_2, , n_m)$ where arguments $n_1, n_2, , n_m \in N$. The function and all its arguments (if exist) are declared as constants in a way that the function calculate and return a value without affecting any numeric state variable.*
- $eff = eff_P \cup eff_N$ is the list of effects.

  $eff_P$ defined over $P$ are positive and negative propositional effects that add or remove litterals.

  $eff_N$ are numeric effects, such that: $\forall e \in eff_N$, $e = (n := g)$, where $n \in N$ and $g$ is an external function or an expression.

A planning problem is defined as:

1. A nonempty state space $S$, which is a finite or countably infinite set of states.
2. For each state $s \in S$, a finite set of applicable actions $A(s)$.
3. A state transition function f that produces a state $s' = f(s,a) \in S$ for every $s \in S$ and $a \in A(s)$.
4. An initial state $s_I \in S$.
5. A set of goal conditions $G$ satisfied in $s_G \in S$.

   The set of goal conditions $G = G_P \cup G_N$, where: $G_P$ defined over $P$ are propositional goals and $G_N$ are numeric goals that should be satisfied in the goal state $s_G$, such that: $\forall l \in G_N$, $l = (n \theta g)$, where $n \in N$, $\theta \in \{<, \leq, =, >, \geq\}$ and $g$ is an external function or an expression.

# 5 PLANNING GRAPH EXTENSION

In this section we present an extension to the planning graph structure to support numeric variables handling and external functions executions. We start the creation of the planning graph by instantiating the propositional variables of all the actions of the planning problem. This instantiation is done once before the graph expansion by substituting the propositional variables by all the combinations of the world objects. Therefore, none of numeric variables are substituted before the planning graph expansion process.// We initialize the fact level 0 to the initial state of the planning problem then we loop while the goal is not satisfied at the current fact level which is initially the level 0. For each one of the previously instantiated actions, we test first if all the propositional preconditions $pre_P$ exist in the current level. If the first test is successful then we proceed to test the numerical conditions $pre_N$ otherwise, we skip this second test. If the numerical preconditions $pre_N$ of the action are satisfied in the current level we add this action to the applicable actions list of the current action level. For

rotate(Position p1, Position p2, robot r)
pre: $p1 \neq p2$, $at(r, p1)$, fcos(cont short $\theta$) $\prec 1$
eff: $at(r, p2)$, $\neg at(r, p1)$, $\theta$ :=frotation(const short $\theta$)

Figure 1: An action with external functions.

example, in the action rotate (fig.1) the first condition $p1 \neq p2$ is tested at the propositional instantiation of the actions as the condition concerns only world objects which are not covered by a relation (not a literal) or an external function. This kind of tests avoids instantiating non useful actions from the beginning. The second condition $at(r, p1)$ is a literal so it belongs to the propositional preconditions $pre_P$ and it is tested first for choosing the applicable actions at the current level. Suppose that $at(r, p1)$ exists in the current fact level thus, the algorithm will proceed by testing the external function "fcos()" for the value of the formal parameter $\theta$ passed from the current fact level. If the value returned by "fcos()" is less than 1, then this ac-

tion will be added to the applicable actions of the current action level.

At each action level, the numeric variables of an action are instantiated just before the action application at this current level. This instantiation is done by replacing the variables by the corresponding values from the current fact level. This technique of "'on the fly"' instantiation allows avoiding the flood of numeric values and can be used for continuous and discrete numeric values. Only the variables that are parameters of external functions and the right hand side numerical variables of expressions are instantiated from the current fact level. All these on the fly instantiated variables are added implicitly to the action preconditions by direct edges. These edges to implicit preconditions will allow us to follow the trace of the functions application within the planning graph during the extraction process. We run through the planning graph during the extraction of a plan without making difference between implicit and explicit preconditions. This technique allows us to use black box functions because we are only concerned by the parameters and the returned value of a function to follow its trace through edges during the planning process. After all actions are instantiated propositionally and numerically at the current level, the application of these actions can take place.

The application of actions at the current action level will lead to the creation of the next fact level. Positive propositional effects in $eff_P$ of an applicable action $a$ are pointed as added by $a$ to the next fact level and negative effects are pointed as deleted by $a$ from the current fact level using edges. For each numerical variable which is assigned to an external function or to an expression at the numerical effects in $eff_N$, the old values of these variables are pointed as deleted by $a$ from the current fact level and the new values returned by functions or expressions are pointed as added by $a$ to the next current level. This organization of numerical effects will allow a uniform representation for all types of values and thus, the run through the graph will not be affected by the function results. From now on, the next fact level becomes the current fact level and the loop continue until the goal conditions are found at this current fact level or until a sufficient condition indicates that the problem has no solution.

# 6 CONCLUSION

In this paper we proposed an extension to the planning graph construction algorithm in order to allow it supporting the application of functions written in a given programming language. This extension allows the application of functions in a planning graph irrespective of their contents. As the planning graph is build in a forward pass the parameters and the results of the functions can be registered consequently during the graph construction process. During the extraction of a plan in a backward pass the algorithm is only concerned by the graph nodes from which it finds a solution by following the direct edges of the graph. We have proposed a method to instantiate numeric variables dynamically, in a way that the instantiation of these variables is done step by step during the graph expansion process. This technique resolves the problem of the flood that can be generated by the numeric domains and allows handling continuous and discrete numerical domains during a planning task.

# REFERENCES


A., G. and D., L. (2005). Plan constraints and preferences for PDDL3. Technical Report Technical report, R.T. 2005-08-07, Dept. of Electronics for Automation, U. of Brescia, Brescia, Italy.

BAK M., P. N. and O., R. (2000). Path following mobile robot in the presence of velocity constraints. Technical report, Kongens Lyngby, Denmark : Technical University of Denmark.

Blum, A. L. and Furst, M. L. (1995). Fast planning through planning graph analysis. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI95)*, pages 1636–1642.

D., S. and D., W. (1999). Temporal planning with mutual exclusion reasoning. *16th International Joint Conference on Artificial Intelligence (IJCAI99)*.

Do, M. B. and Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporel planner. *In Proceedings of the European Conference on Planning (ECP 2001)*.

Geffner, H. (1999). Functional strips: A more flexible language for planning and problem solving. *Logic-based AI Workshop, Washington D.C.*

Hoffman, J. (2001). FF: The fast-forward planning system. *AI Magazine*, 22:57 – 62.

Hoffmann, J. (2002). Extending FF to numerical state variables. *In Proceedings of the 15th European Conference on Artificial Intelligence, Lyon, France*.

Schmid, U., Müller, M., and Wysotzki, F. (2000). Integrating function application in state-based planning. *Submitted manuscript*.

Smith, D. and Weld, D. (1999). Temporal planning with mutual exclusion reasoning. *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*.