A VISUALIZATION ENVIRONMENT FOR PLANNING

DIMITRIS VRAKAS

Dept. Of Informatics, Aristotle University of Thessaloniki,
Thessaloniki, 54124, Greece
dvrakas@csd.auth.gr

IOANNIS VLAHAVAS

Dept. Of Informatics, Aristotle University of Thessaloniki,
Thessaloniki, 54124, Greece
vlahavas@csd.auth.gr

This article presents ViTAPlan-2, a visual tool for adaptive planning that is build on top of HAPRC, a rule-configurable planning system, which automatically adapts to each problem, in order to achieve best performance. Apart from HAPRC, ViTAPlan can be interfaced with any other planning system that supports the PDDL language. More than just being a user friendly environment for executing the underlying planner, the tool serves as a unified planning environment for encoding a new problem problem, solving it, visualizing the solution and monitoring its execution on a simulation of the problem's word. The tool consists of various sub-systems, each one accompanied by a graphical interface, that collaborate with each other and assist the user, whether he is a knowledge engineer, a domain expert, an academic or even an end user in industry, to carry out complex planning tasks.

Keywords: Automated Planning; Graphical Interfaces; Machine Learning; Knowledge Engineering

1. Introduction

Automated Planning has been an active research topic for almost 40 years and during this period a great number of papers describing new methods, techniques and systems have been presented that mainly focus on ways to improve the efficiency of planning systems. However, there are not many successful examples of planning systems adapting to industrial use. From a technical point of view, this can be mainly explained by four reasons: a) There is a general disbelief of managers and workers in industry that AI tools can really assist them, b) There is a need for systems that combine methods from many areas of AI, such as Planning, Scheduling and Optimization, c) The industry needs more sophisticated algorithms than can scale up to solve real-world problems and d) In order for workers in industry to make use of these intelligent systems, they must be equipped with user friendly interfaces that: i) allow the user to intervene in certain points and ii) can reason about the provided solution.

The greater problems that one faces when he tries to contact companies and organizations for installing a planning system, come from the workers themselves. These problems concern two issues: a) It has been noticed that people find it hard to trust automated systems when it comes for crucial processes. There are a lot of people who still think that they can do better than machines. b) The lack of information is the cause of a quite widespread phobia towards computers and automated machines. There are still a lot of people who think that they will be replaced or governed by machines and the try to defend their posts by rejecting everything new.

Although it is necessary for researchers to specialize in very specific parts of their research area, commercial systems, dealing with real time problems, have to combine techniques and methods from many areas. It has been shown that AI Planning techniques for example, are inadequate to face with the complexity and the generality of real world problems. For example, it has been proven that Scheduling and Constraint solving techniques can handle resources more efficiently. Commercial applications must combine methods from many areas of AI and probably from other areas of computer science as well.

Another issue that must be dealt is the large gap between toy problems used by researcher for developing and testing their algorithms and the actual problems faced be people in industry. Researchers are usually unaware of the size of real world problems or they simplify these problems in order to handle with them. But these algorithms prove themselves inadequate to be adopted by commercial software. So it is a general conclusion that researcher should start dealing with more realistic problems. This research direction was also given during the last AIPS Planning competitions where there was a tendency to test planners on problems closer to reality.

Last but certainly not least is the direct need of software based on AI tools to be accompanied with user friendly interfaces. Since the user will be a manager or even a simple worker in a company and not a computer scientist, the software must be easy to use. Furthermore, it must enable the user to intervene in certain points for two reasons: a) Mixed initiative systems can deal with real-world problems better and b) people in companies do not like to take commands and therefore a black box which can not reason about its output will not do. So it is necessary for the software to cooperate with the user in the process of solving the problem in hand, since people have a more abstract model of the problem in their mind and are better in improvising, while computers can more efficiently deal with lower levels of the problem.

This article describes ViTAPlan-2, a visual tool for adaptive planning, which is equipped with a rule system able to automatically fine-tune the planner based on the morphology of the problem in hand. The tool has been developed for the HAP planning system, but it can be used as a graphical platform for any other modern planner that supports the PDDL language. The tool consists of various sub-systems that collaborate in order to carry out several planning tasks and provides the user with a large number of functionalities that are of interest to both industry and academia.

The rest of the article is organized as follows: The next section presents an overview of the work related to automated planning systems and graphical environments for planning. Section 3 describes the architecture of the visual tool and briefly describes the contained sub-systems. Sections 4 and 5 present the execution module that interfaces the visual tool with the planning system and the knowledge module that is responsible for the automatic configuration of the planning parameters respectively. Section 6 analyzes the graphical tool for visualizing and designing problems and domains and illustrates the use of the sub-system with concrete examples. The next section presents the two sub-systems that visualize the plans and simulate their execution in virtual worlds and finally section 8 concludes the article and poses future directions.

2. Related Work

Two of the most promising trends in building fast domain-independent planning systems were presented over the last few years.

The first one consists of the transformation of the classical search in the space of states to other kinds of problems, which can be solved more easily. Examples of this category are the SATPLAN[13] and BLACKBOX[14] planning system, the evolutionary GRAPHPLAN[1] and certain extensions of GRAPHPLAN as the famous STAN[17] planner.

SATPLAN and BLACKBOX transform the planning problem into a satisfiability problem, which consists of a number of boolean variables and certain clauses between these variables. The goal of the problem is to assign values to the variables in such a way that establishes all of the clauses.
GRAPHPLAN on the other hand creates a concrete structure, called the planning graph, where the nodes correspond to facts of the domain and edges to actions that either achieve or delete these facts. Then the planner searches for solutions in the planning graph. GRAPHPLAN has the ability to produce parallel plans, where the number of steps is guaranteed to be minimum.

The second category is based on a relatively simple idea where a general domain independent heuristic function is embodied in a heuristic search algorithm such as Hill Climbing, Best-First Search or A*. Examples of planning systems in this category are the ASP/HSP family[2], AltAlt[19], FF[11], YAHSP[22] and Macro FF[3].

The planners of the latter category rely on the same idea to construct their heuristic function. They relax the planning problem by ignoring the delete lists of the domain operators and starting either from the Initial State or the Goals they construct a leveled graph of facts, noting for every fact f the level at which it was achieved L(f). In order to evaluate a state S, the heuristic function takes into account the values of L(f) for each $f \in S$.

The systems presented above are examples of real fast planners that are able to scale up to quite difficult problems. However, there are still open issues to be addressed that are crucial to industry, such as temporal planning or efficient handling of resources. Although there has been an effort during the last few years to deal with these issues there is still only a small number of systems capable of solving near real world problems.

An example of this trend is the SGPlan system[5], which won the won the first prize in the suboptimal temporal metric track and a second prize in the suboptimal propositional track in the Fourth International Planning Competition (IPC4). The basic idea behind SGPlan is to partition problem constraints by their subgoals into multiple subsets, solve each subproblem individually, and resolve inconsistent global constraints across subproblems based on a penalty formulation.

Another system able to handle planning problems that incorporate the notion of time and consumable resources is the LPG-TD planner[9], which is an extension of the LPG planning system. Like the previous version of LPG, the new version is based on a stochastic local search in the space of particular “action graphs” derived from the planning problem specification. In LPG-TD, this graph representation has been extended to deal with the new features of PDDL2.2, as well to improve the management of durative actions and of numerical expressions.

There are also some older systems that combine planning with constraint satisfaction techniques in order to deal with complex problems with time and constraints. Such systems include the Metric FF Planner[12], the S-MEP[20] and the SPN Neural Planning Methodology[4].

As far as user interfaces are concerned, there have been several approaches from institutes and researchers to create visual tools for defining problems and running planning systems, such as the GIPO system[18], the SIPE-2[29] and the ASPEN[1] graphical user interfaces. Moreover, there is a number of approaches in building visual interfaces for specific applications of planning. The PacoPlan project [24] aims in building a web-based planning interface for specific domains. AsbruView[15] is a visual user interface for time-oriented skeletal plans representing complex medical procedures. Another example of visual interfaces for planning is the work of the MAPLE research group at the university of Maryland[16], which concerns the implementations of a 3D graphical interface for representing hierarchical plans with many levels of abstractions and interactions among the parts of the plan. Although these approaches are very interesting and provide the community with useful tools for planning, there is still a lot of work to be done in order to create an integrated system that meets the needs of the potential users.

3. Tool's Architecture

The visual planning environment is based on the first version of ViTAPlan[27,28], which has been extended in several ways. The architecture of the tool, which is outlined in Figure 1, consists of the following sub-systems, which are discussed in more detail later in the article: a) designing, b) configuration, c) solving, d) visualizing and e) simulating.

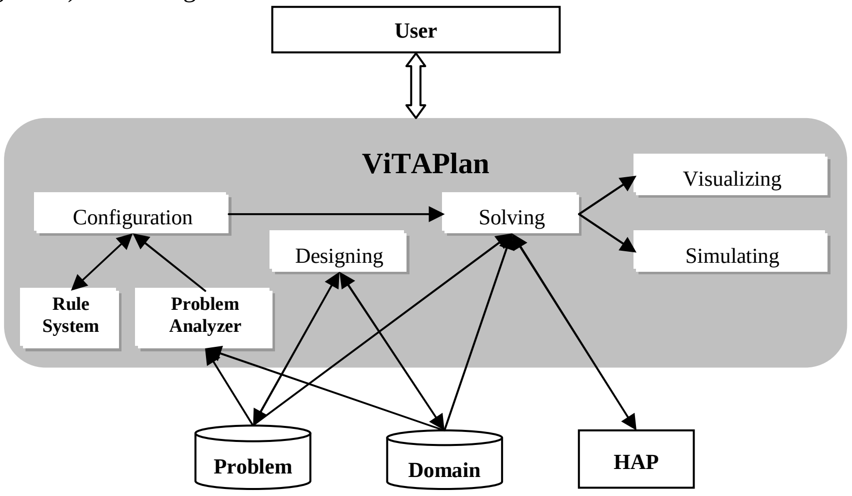


Figure 1. ViTAPlan's Architecture

The designing module provides visual representations of planning domains and problems through graphs that assist the user in comprehending their structure. Furthermore, the user is provided with graphical elements and tools for designing new domains and new problems of existing domains. This module communicates with the file system of the operating system, in order to save the designs in a planner – readable format (i.e. PDDL[10] planning language) and also load and visualize past domains and problems.

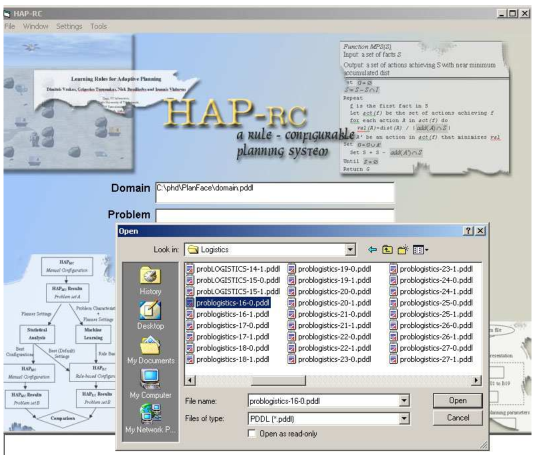
The configuration module of ViTAPlan-2 deals with the automatic fine-tuning of the planning parameters. This task is performed through two steps that are implemented in different sub-systems. The first one, called Problem Analyzer, reads the description of the problem from the input files, analyzes it and outputs a vector of numbers that correspond to the values of 35 measurable attributes that quantify the morphology of the problem in hand. The rule system, which is consulted after the analysis of the problem, contains a number of rules that associate specific values or value ranges of the problems' attributes with configurations of the planning parameters that guarantee good performance.

The solving module inputs the description of the problems (PDDL files of the domain and the problem) along with the values of the planner's parameters and executes the planner (HAP or any other system attached to ViTAPlan) in order to obtain a solution (plan) to the problem. The planner's parameters can be adjusted either by hand or automatically via the configuration module.

The last two sub-systems (visualization and simulation) present the plan that was inputted from the execution module in several forms. The visualization module presents the plan as a directed graph, through which the user can identify the positive and negative interactions among the actions and experiment with different orderings in which the plan's steps should be executed. The second module simulates the problem's world presenting all the intermediate states that occur after the execution of each action.

4. Interface for Planning Systems

The main feature of ViTAPlan is to allow the use of the underlying planning system in a friendlier and more accurate way. This interface between the user and the planning engine is carried out by the solving module of the visual environment. The inputs that must be supplied to the planning system are: a) the domain file, b) the problem file and c) the values of the planning parameters, in case of adjustable planners.



Domain: C:\phd\PlanFace\domain.pddl

Problem:

Look in: Logistics

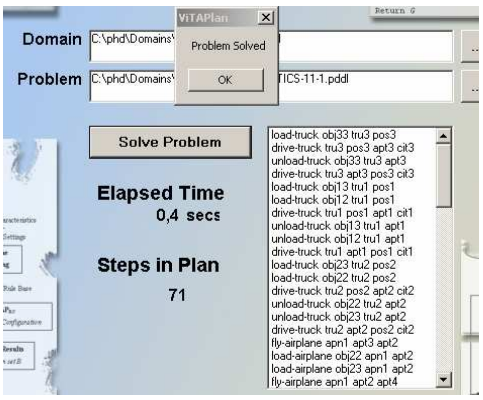
probLOGISTICS-14-1.pddl	problogistics-19-0.pddl	problogistics-23-1.pddl
probLOGISTICS-15-0.pddl	problogistics-19-1.pddl	problogistics-24-0.pddl
probLOGISTICS-15-1.pddl	problogistics-20-0.pddl	problogistics-24-1.pddl
problogistics-16-0.pddl	problogistics-20-1.pddl	problogistics-25-0.pddl
problogistics-16-1.pddl	problogistics-21-0.pddl	problogistics-25-1.pddl
problogistics-17-0.pddl	problogistics-21-1.pddl	problogistics-26-0.pddl
problogistics-17-1.pddl	problogistics-22-0.pddl	problogistics-26-1.pddl
problogistics-18-0.pddl	problogistics-22-1.pddl	problogistics-27-0.pddl
problogistics-18-1.pddl	problogistics-23-0.pddl	problogistics-27-1.pddl

File name: problogistics-16-0.pddl

Files of type: PDDL (*.pddl)

Figure 2. Selecting the input files

From the initial screen of the interface, which is shown in Figure 2, the user uses common dialogues and graphical elements in order to browse for the domain and problem files that will be inputted to the planner. From the same screen the user can also execute the planner and obtain the solution (plan), among with statistics concerning the execution, such as the planning time and the length of the plan. The way the results of the planner are presented to the user is shown in Figure 3.



Domain: C:\phd\Domains\

Problem: C:\phd\Domains\ ... TICS-11-1.pddl

Problem Solved

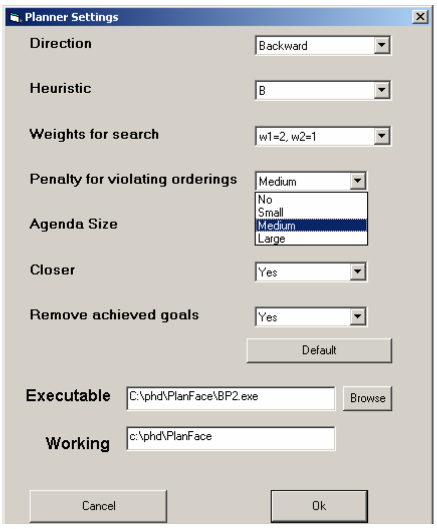
Elapsed Time
0,4 secs

Steps in Plan
71

```
load-truck obj33 tru3 pos3
drive-truck tru3 pos3 apt3 cit3
unload-truck obj33 tru3 apt3
drive-truck tru3 apt3 pos3 cit3
load-truck obj13 tru1 pos1
load-truck obj12 tru1 pos1
drive-truck tru1 pos1 apt1 cit1
unload-truck obj13 tru1 apt1
unload-truck obj12 tru1 apt1
drive-truck tru1 apt1 pos1 cit1
load-truck obj23 tru2 pos2
load-truck obj22 tru2 pos2
drive-truck tru2 pos2 apt2 cit2
unload-truck obj22 tru2 apt2
unload-truck obj23 tru2 apt2
drive-truck tru2 apt2 pos2 cit2
fly-airplane apn1 apt3 apt2
load-airplane obj22 apn1 apt2
load-airplane obj23 apn1 apt2
fly-airplane apn1 apt2 apt4
```

Figure 3. Solving the problem

There are three ways for tuning the planner's parameters in ViTAPlan: a) using the default values provided by the system, b) let the user assign the values by hand or c) use the configuration module in order to have the parameters set automatically. The user can select among the first two ways through the settings window presented in Figure 4. The parameters presented in this window correspond to the HAP planner which is embodied in ViTAPlan.



Direction	Backward
Heuristic	B
Weights for search	w1=2, w2=1
Penalty for violating orderings	Medium (No / Small / Medium / Large)
Agenda Size	
Closer	Yes
Remove achieved goals	Yes
Executable	C:\phd\PlanFace\BP2.exe
Working	c:\phd\PlanFace

Figure 4. Setting the planning parameters

4.1. Hap planner

HAP, is a highly adjustable planning system that can be customized by the user through a number of parameters. These parameters concern the type of search, the quality of the heuristic and several other features that affect the planning process. The HAP system is based on the BP (Bi-directional Planner) planning system[23] and uses an extended version of the ACE[25] (Action Evaluation) heuristic.

HAP is capable of planning in both directions (progression and regression). The system is quite symmetric and for each critical part of the planner, e.g. calculation of mutexes, discovery of goal orderings, computation of the heuristic, search strategies etc., there are implementations for both directions. The *direction* of search is the first adjustable parameter of HAP used in tests, with the following values: a) 0 (Regression or Backward chaining) and b) 1 (Progression or Forward chaining).

As for the search itself, HAP adopts a weighted A* strategy with two independent weights: w_1 for the estimated cost for reaching the final state and w_2 for the accumulated cost of reaching the current state from the starting state (initial or goals depending on the selected direction). For the tests with HAP, we used four different assignments for the variable *weights* which correspond to different assignments for w_1 and w_2: a) 0 (w_1 =1, w_2 =0), b) 1 (w_1 =3, w_2 =1), c) 2 (w_1 =2, w_2 =1) and d) 3 (w_1 =1, w_2 =1).

The size of the planning agenda (denoted as *sof_agenda*) of HAP also affects the search strategy and it can also be set by the user. For example, if we set *sof_agenda* to 1 and w_2 to 0, the search algorithm becomes pure Hill-Climbing, while by setting *sof_agenda* to 1, w_1 to 1 and w_2 to 1 the search algorithm becomes A*. Generally, by increasing the size of the agenda we reduce the risk of not finding a solution, even if at least one exists, while by reducing the size of the agenda the search algorithm becomes faster and we ensure that the planner will not run out of memory. For the tests we used three different settings for the size of the agenda: a) 1, b) 100 and c) 1000

The *OB* and *OB-R* functions introduced in BP and ACE respectively, are also adopted by HAP in order to search the states of the search for violations of orderings between the facts of either the initial state or the goals, depending on the direction of the search. For each violation contained in a state, the estimated value of this state that is returned by the heuristic function, is increased by violation penalty, which is a constant number supplied by the user. For the experiments of this work we tested the HAP system with three different values of *violation_penalty*: a) 0, b) 10 and c) 100.

The HAP system employs the heuristic function of the ACE planner, plus two variations of it, which are in general more fine-grained. There are implementations of the heuristic functions for both planning directions. All the heuristic functions are constructed in a pre-planning phase by performing a relaxed search in the opposite direction of the one used in the search phase. During this relaxed search the heuristic function computes estimations for the distances of all grounded actions of the problem.

The user may select the heuristic function by configuring the *heuristic_order* parameter. The three acceptable values are: a) 1 for the initial heuristic, b) 2 for the first variation and c) 3 for the second variation.

HAP also embodies a technique for simplifying the definition of the sub-problem in hand. This technique eliminates from the definition of the sub-problem (current state and goals) all the goals that have already been achieved in the current state and do not interfere in any way with the achievement of the remaining goals. In order to do this the techniques performs, off-line before the search process, a dependency analysis on the goals of the problem. The parameter *remove_subgoals* is used to turn on (value 1) and off (value 0) this feature of the planning system.

The last parameter of HAP is *equal_estimation*, which defines the way in which states with the same estimated distances are treated. If *equal_estimation* is set to 0 then between two states with the same value in the heuristic function, the one with the largest distance from the starting state (number of actions applied so far) is preferred. If *equal_estimation* is set to 1, then the search strategy will prefer the state, which is closer to the starting state.

4.2. Embedding other planning systems

The current version of ViTAPlan embodies the HAP system, but the environment is open and the user can easily attach any other planner that reads the PDDL[10] language. The communication protocol between the visual environment and the planner (including HAP) is outlined in Figure 5.

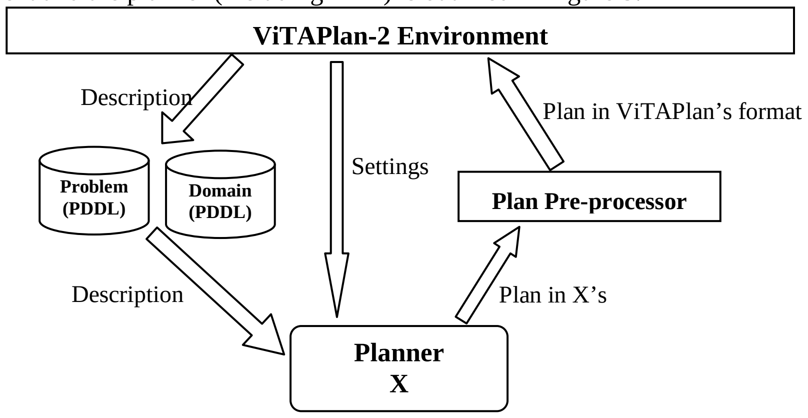


Figure 5. Communication with the planner

As already discussed the data that should be transmitted between ViTAPlan and the planning system are: a) the description of the domain and the problem, b) optionally the settings for the planner's parameters and c) the plan that solves the problem.

Concerning the description of the problem, ViTAPlan is able to extract PDDL files, which is the standard definition language for all modern planners. Therefore it is trivial to submit the problem to any new planning system.

For each configurable parameter, ViTAPlan needs a description, the option used to set this parameter in the planner and the domain of values. This information must be specified by the user in order to add a new planner in the environment. The current version of ViTAPlan supports only discrete values for the domains of the parameters, but this can very easily be extended to support continuous values or other data types (e.g. booleans). For example, Table 1 presents part of the information that should be inputted to ViTAPlan-2 in order to connect it with the LPG planning system[8].

Table 1. Specification of LPG's parameters

Parameter	Option	Values
Heuristic identifier	-h	1,2
Max number of restarts	-restarts	1,2,3,4,5,6,7,8,9….
Noise added to Walksat	-noise	0,0.1,0.2,0.3….
...		

The output of any planner to a given problem is a sequence of actions that achieve the predefined goals. However, since there has not been a standard for describing plans yet, each planner may present the sequences of actions in a different way. Therefore a necessary step in order to attach a different planning system *X* on ViTAPlan-2, is to format its output, either by modifying the system, or by adding a pre-processor that reads the plan in *X*'s format and transforms it. Figure 6 presents an example of a plan in ViTAPlan's format.

```
Begin of Plan
1: (drive truck0 distributor1 distributor0)
2: (lift hoist1 crate0 pallet1 distributor0)
3: (load hoist1 crate0 truck0 distributor0)
…
11: (drop hoist1 crate1 pallet1 distributor0)
End of Plan
```

Figure 6. A plan's excerpt in ViTAPlan's format

5. Configuration Module

The Configuration module of ViTAPlan is a sub-system able to adjust the planning parameters in automatic manner. This feature is currently available only for use with the HAP planning system, although the methodology for the automatic configuration of planning parameters is general and can be applied to other planners as well [21].

The automatic configuration is based on HAP-RC[26], which uses a rule system in order to automatically select the best settings for each planning parameter, based on the morphology of the problem in hand. HAP-RC, whose architecture is outlined in Figure 7 is actually HAP with two additional modules (Problem Analyzer and Rule System) which are utilized off-line, just after reading the representation of the problem in order to fine tune the planning parameters of HAP.

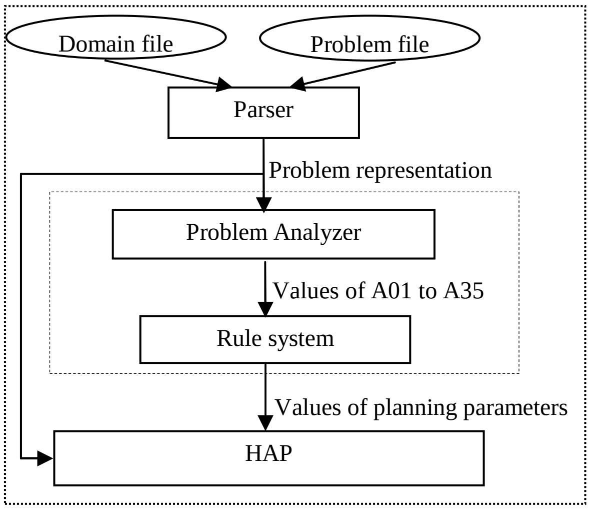


Figure 7. HAP-RC Architecture

The role of the Problem Analyzer is to identify the values of a specific set of 35 problem characteristics (noted as A1 to A35). These characteristics include measurable attributes of planning problems, such as number of facts per predicate and the branching factor of the problem that present the internal structure of the problems in a quantified way. After the identification of the values of the attributes, which may requires a limited search in the problem, the analyzer feeds the Rule system with a vector containing the values for the 35 problem attributes.

The Rule system contains a number of rules of the following format:

If preconditions list Then actions list

The preconditions of the rules check if the values of the problem's attributes comply with some constraints on them, while the actions set one or more planning parameters to specific values. For example, the rule:

If A24<1.8 and A17<9.7 Then direction=1 and closer=yes

will trigger in a given problem if the values of A24 (ratio between the branching factors of the two directions) and A17 (standard deviation of the average number of actions deleting a fact) are smaller than 1.8 and larger than 9.7 respectively. If this rule is eventually fired, then the planning direction will be set to forward and the search algorithm will use the technique for overcoming plateaus.

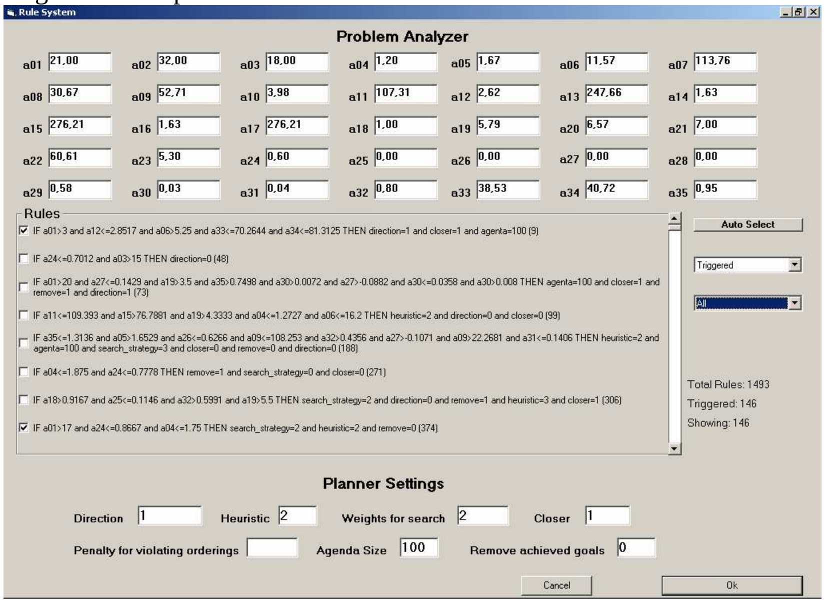
What these rules actually do, is to propose setups for the planning parameters that worked efficiently in similar problems in the past. This knowledge has been extracted from Machine Learning techniques on data produced by thorough experiments with the HAP system. More specifically, we tested all the possible combinations of the parameters of HAP on a large set of problems from various domains and for each run we kept record of the values of the problem attributes, the specific setup for HAP and the value for a metric combining planning time and plan length. The data set was then fed to a Machine Learning tool in order to learn a rule-based classification model that would discriminate between good and bad value of the metric based on the rest of the attributes.

The Configuration module of ViTAPlan-2 also provides the user with the option to use the Problem Analyzer and the Rule System of HAP-RC in order to automatically fine-tune the planning parameters of HAP. The relevant window of the interface is shown in Figure 8. This window is divided in three parts: a) the first part shows the discretized values for the 35 problem characteristics, as produced by the Problem analyzer, b) the second part provides the user with the list of the rules that comprise the core knowledge of the system and c) the last part provides the user with the proposed values for the planning parameters of HAP.

The values of the problem's attributes are presented in order to check for the triggered rules, but more importantly to assist the knowledge engineer or the domain expert in decoding the internal structure of the problem and extract useful insights from it. The tool presents the following information for each one of the 35 attributes: a) the code name (e.g *A07*), b) a description of the attribute (e.g. *Standard deviation of the number of facts per predicate*), c) the arithmetic value, d) a discretized value (e.g. *small*) and e) the usual upper and lower limit for its value.

The rules are shown in the appropriate frame of the environment sorted by decreasing order of their confidence, as this was calculated by the learning algorithm. ViTAPlan presents all the rules, but the user is able to control the viewable part of the rules through two controls that select only the triggered rules or the rules that affect a specific parameter (e.g. heuristic function).

From the set of triggered rules, ViTAPlan-2 makes an initial choice, selecting the rules with the highest confidence factor that are not in conflict with any other already selected rule. We say that two rules are in conflict, if they propose different setups for the same parameter. The configuration module uses the initial subset of selected rules in order to calculate the values of the parameters and present them to the user, leaving each unset parameter to its default value.



a01	21,00	a02	32,00	a03	18,00	a04	1,20	a05	1,67	a06	11,57	a07	113,76
a08	30,67	a09	52,71	a10	3,98	a11	107,31	a12	2,62	a13	247,66	a14	1,63
a15	276,21	a16	1,63	a17	276,21	a18	1,00	a19	5,79	a20	6,57	a21	7,00
a22	60,61	a23	5,30	a24	0,60	a25	0,00	a26	0,00	a27	0,00	a28	0,00
a29	0,58	a30	0,03	a31	0,04	a32	0,80	a33	38,53	a34	40,72	a35	0,95

- [✓] IF a01>3 and a12<=2.8517 and a06>5.25 and a33<=70.2644 and a34<=81.3125 THEN direction=1 and closer=1 and agenta=100 (9)
- [ ] IF a24<=0.7012 and a03>15 THEN direction=0 (48)
- [ ] IF a01>20 and a27<=0.1429 and a19>3.5 and a35>0.7498 and a30>0.0072 and a27>-0.0882 and a30<=0.0358 and a30>0.008 THEN agenta=100 and closer=1 and remove=1 and direction=1 (73)
- [ ] IF a11<=109.393 and a15>76.7881 and a19>4.3333 and a04<=1.2727 and a06<=16.2 THEN heuristic=2 and direction=0 and closer=0 (99)
- [ ] IF a35<=1.3136 and a05>1.6529 and a26<=0.6266 and a09<=108.253 and a32>0.4356 and a27>-0.1071 and a09>22.2681 and a31<=0.1406 THEN heuristic=2 and agenta=100 and search_strategy=3 and closer=0 and remove=0 and direction=0 (188)
- [ ] IF a04<=1.875 and a24<=0.7778 THEN remove=1 and search_strategy=0 and closer=0 (271)
- [ ] IF a18>0.9167 and a25<=0.1146 and a32>0.5991 and a19>5.5 THEN search_strategy=2 and direction=0 and remove=1 and heuristic=3 and closer=1 (306)
- [✓] IF a01>17 and a24<=0.8667 and a04<=1.75 THEN search_strategy=2 and heuristic=2 and remove=0 (374)

Total Rules: 1493
Triggered: 146
Showing: 146

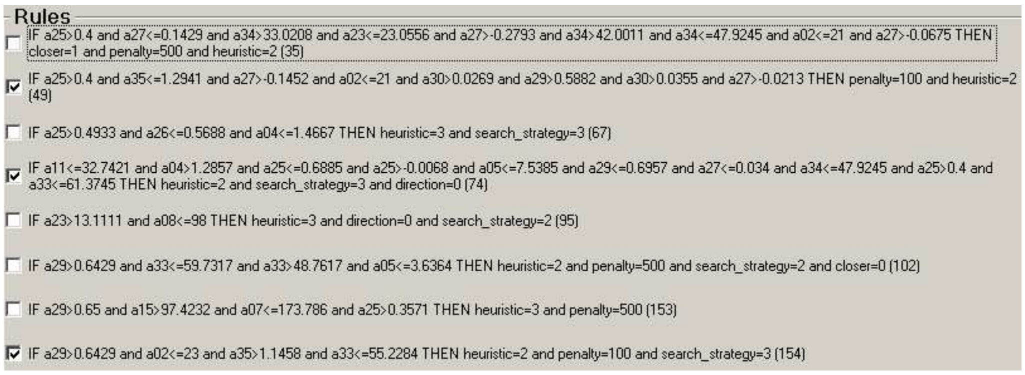
Planner Settings

Direction	1	Heuristic	2	Weights for search	2	Closer	1
Penalty for violating orderings		Agenda Size	100	Remove achieved goals	0		

Figure 8. The rule system

Instead of just accepting or rejecting the proposed parameter setting, the user has the ability to interfere with the rule system, modifying so the adopted conflict resolution strategy. More specifically, the user may either include a new rule in the firing set, or request the removal of a selected one and thus alter the firing set and therefore the setup of the parameters. Each time the user request the firing of a new rule R_i, the module automatically checks the resulting firing set for conflicts and removes all the rules that propose contradictory values for the parameters affected by R_i.

Consider for example the case in Figure 9, which presents a portion of the triggered rules, i.e. those that require values for the attributes A01 to A35 that are compatible with the problem in hand. The initial selection contains rules 49, 74 and 154 that define the values 100, 2, 3 and 0 for the parameters *penalty*, *heuristic*, *search_strategy* and *direction* respectively.


Rules

☐ IF a25>0.4 and a27<=0.1429 and a34>33.0208 and a23<=23.0556 and a27>-0.2793 and a34>42.0011 and a34<=47.9245 and a02<=21 and a27>-0.0675 THEN closer=1 and penalty=500 and heuristic=2 (35)

☑ IF a25>0.4 and a35<=1.2941 and a27>-0.1452 and a02<=21 and a30>0.0269 and a29>0.5882 and a30>0.0355 and a27>-0.0213 THEN penalty=100 and heuristic=2 (49)

☐ IF a25>0.4933 and a26<=0.5688 and a04<=1.4667 THEN heuristic=3 and search_strategy=3 (67)

☑ IF a11<=32.7421 and a04>1.2857 and a25<=0.6885 and a25>-0.0068 and a05<=7.5385 and a29<=0.6957 and a27<=0.034 and a34<=47.9245 and a25>0.4 and a33<=61.3745 THEN heuristic=2 and search_strategy=3 and direction=0 (74)

☐ IF a23>13.1111 and a08<=98 THEN heuristic=3 and direction=0 and search_strategy=2 (95)

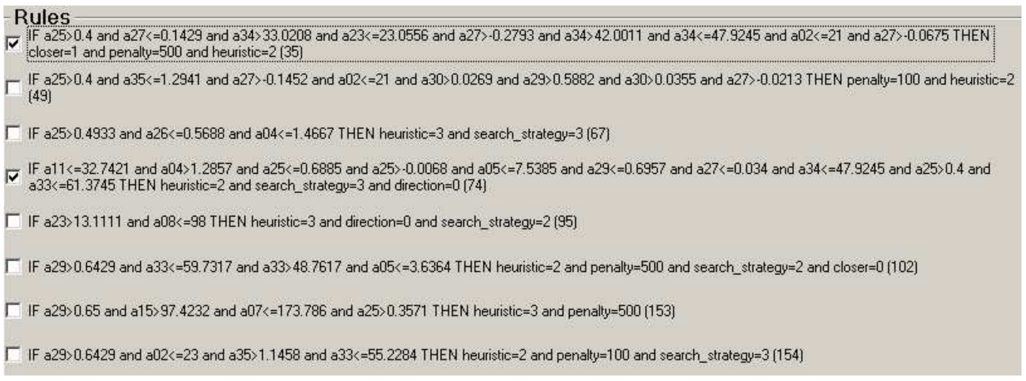
☐ IF a29>0.6429 and a33<=59.7317 and a33>48.7617 and a05<=3.6364 THEN heuristic=2 and penalty=500 and search_strategy=2 and closer=0 (102)

☐ IF a29>0.65 and a15>97.4232 and a07<=173.786 and a25>0.3571 THEN heuristic=3 and penalty=500 (153)

☑ IF a29>0.6429 and a02<=23 and a35>1.1458 and a33<=55.2284 THEN heuristic=2 and penalty=100 and search_strategy=3 (154)

Figure 9. The initial firing set

Lets also suppose that the user requests rule number 35 (the first rule in Figure 9) to be included in the firing set. Rule 35 is in contradiction with rule 49, since the first sets *penalty* to 500 while the second sets it to 100 and with rule 154 for the same reason. There is no contradiction with rule 74, since the only common parameter is *heuristic* and they propose the same value (2) for it. Therefore after the inclusion of rule 35, rules 49 and 154 are removed form the firing set as shown in Figure 10. The proposed setup of the planner's parameters becomes the following: *closer=1*, *penalty=500*, *heuristic=2*, *search_strategy=3* and *direction=0*.



Rules

```
[✓] IF a25>0.4 and a27<=0.1429 and a34>33.0208 and a23<=23.0556 and a27>-0.2793 and a34>42.0011 and a34<=47.9245 and a02<=21 and a27>-0.0675 THEN
    closer=1 and penalty=500 and heuristic=2 (35)
[ ] IF a25>0.4 and a35<=1.2941 and a27>-0.1452 and a02<=21 and a30>0.0269 and a29>0.5882 and a30>0.0355 and a27>-0.0213 THEN penalty=100 and heuristic=2
    (49)
[ ] IF a25>0.4933 and a26<=0.5688 and a04<=1.4667 THEN heuristic=3 and search_strategy=3 (67)
[✓] IF a11<=32.7421 and a04>1.2857 and a25<=0.6885 and a25>-0.0068 and a05<=7.5385 and a29<=0.6957 and a27<=0.034 and a34<=47.9245 and a25>0.4 and
    a33<=61.3745 THEN heuristic=2 and search_strategy=3 and direction=0 (74)
[ ] IF a23>13.1111 and a08<=98 THEN heuristic=3 and direction=0 and search_strategy=2 (95)
[ ] IF a29>0.6429 and a33<=59.7317 and a33>48.7617 and a05<=3.6364 THEN heuristic=2 and penalty=500 and search_strategy=2 and closer=0 (102)
[ ] IF a29>0.65 and a15>97.4232 and a07<=173.786 and a25>0.3571 THEN heuristic=3 and penalty=500 (153)
[ ] IF a29>0.6429 and a02<=23 and a35>1.1458 and a33<=55.2284 THEN heuristic=2 and penalty=100 and search_strategy=3 (154)
```

Figure 10. The final firing set

6. Designing Module

The designing module of ViTAPlan-2 is able to present graphical visualizations of planning domains and specific problems that assist the users in better comprehension of their structure. The environment also enables the user to modify existing domains and problems and even create new using pre-designed visual elements and simple mouse movements.

6.1. Visualization of domains

The graphical visualization of planning domains that is adopted by ViTAPlan-2 consists of the entities-relations diagram and the definition of the operators.

6.1.1. Entities-relations diagram

The entities-relations diagram is a directed graph containing two types of nodes and one type of arcs connecting the nodes. The first type of nodes, called entity, is represented in the design as a circle and corresponds to an object class of the domain. The second type is presented by the visual tool as a parallelogram and is used to specify relations between the domain’s classes. The edges connect rectangular with circular nodes and are used to specify which classes take part in each relation.

Consider the gripper domain for example, where there is a robot with *N* grippers that moves in a space, composed of *K* rooms that are all connected with each other. All the rooms are modeled as points and there are connections between each pair of points and therefore the robot is able to reach all rooms starting from any one of them with a simple movement. In the gripper domain there are *L* numbered balls which the robot must carry from their initial position to their destination.

The diagram for the *Gripper* domain, which was used in the AIPS-98 planning competition, is illustrated in Figure 11. There are three object classes in the domain, namely *room*, *ball* and *gripper* that are represented with circles. There is no class defined for the robot, since the domain assumes the presence of only one instance of it and therefore there is no need for an explicit definition.

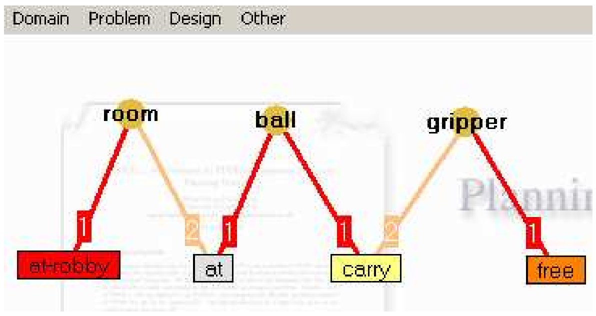


Figure 11. Entities-relations diagram for *gripper*

The domain has four predicates: a) *at-robby*, which specifies the position of the robot and it is connected only with one instance of *room*, b) *at* which specifies the room in which each ball resides and therefore is connected with an instance of *ball* and an instance of *room*, c) *carry* which defines the alternative position of a ball, i.e it is hold by the robot and therefore it is connected with an instance of *ball* and an instance of *gripper* and d) *free* which is connected only with an instance of *gripper* and states that the current gripper does not hold any ball.

Note here that although PDDL, requires only the arity for each predicate and not the type of objects for the arguments, the interface obliges the user to connect each predicate with specific object classes and this is used for the consistency check of the domain design. According to the design of Figure 11, the arity of predicate *holding*, for example, is two and the specific predicate can only be connected with one object of class *ball* and one object of class *gripper*.

6.1.2. Operators

The definition of operators in ViTAPlan follows a declarative schema, which is different from the classical STRIPS approach, although there is a direct way to transform definitions from one approach to the other. More specifically, an operator in the visual environment is represented with two lists, namely *Pre* and *Post*, that contain: a) the facts that must hold in a state S_1 in order to apply the operator and b) the facts that will be true in state S_2 which will result from the application of the operator on S_1. The relations between these lists and the three lists (*Prec*, *Add*, *Del*) of the STRIPS notation are the following:

From STRIPS to ViTAPlan:

$$Pre(A) = Prec(A)$$
$$Post(A) = Add(A) \cup (Prec(A) - Del(A))$$

From ViTAPlan to STRIPS:

$$Prec(A) = Pre(A)$$
$$Add(A) = Post(A) - Pre(A)$$

$$Del(A) = Pre(A) - Post(A)$$

Each operator, in the interface, is represented with a labeled frame, which contains a column of object classes in the middle, two columns of predicates at the two sides of it and connections between the object classes and the predicates. The un-grounded facts that are generated by the classes and the predicates in the left column define the *Pre* list of the operator while the *Post* list is defined by the predicates in the right column.

For example, in the gripper domain there are three operators: a) *move* which allows the robot to move between rooms, b) *pick* which is used in order to lift a ball using a gripper and c) *drop* which is the direct opposite of pick and is used to leave a ball on the ground

The *move* operator is related with two instance of the room class (*room1* and *room2*) which correspond to the initial and the destination room of the robot's move. The *Pre* and *Post* lists of the operator contain only one instance of the *at-robby* relation. In the *Pre* list the at-robby is connected to *room1*, while the latter is replaced by *room2* in the *Post* list. The definition of the move operator is presented in Figure 12.

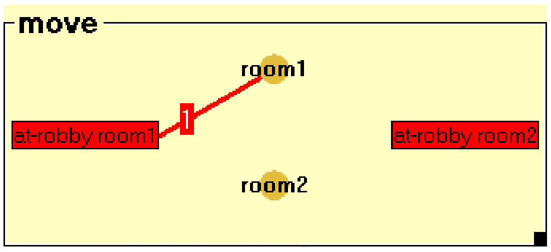


Figure 12. The move operator

The second operator (*pick*), which is presented in Figure 13, contains one instance from three entities, namely *ball1*, *room1* and *gripper1* that correspond to the ball that resided on the room and was picked by a robot's gripper. The *Pre* list defines that both the ball and the robot must be in the same room and that the gripper must be free. The new fact that is contained in the *Post* list is that the gripper holds the ball, while the freedom of the gripper and the fact that the ball resides on the room are deleted. The fact that the robot is in *room1* is contained in both lists (*Pre* and *Post*) since it is not deleted.

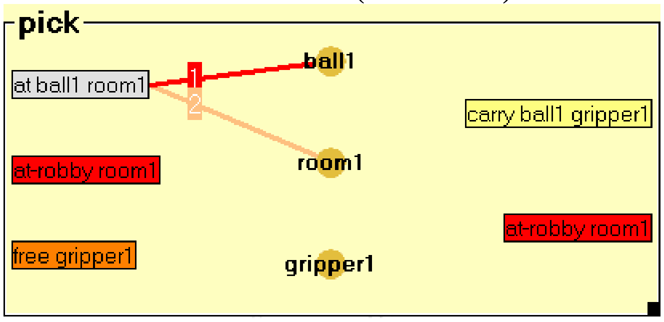


Figure 13. The pick operator

Similarly we define the *drop* operator which is presented in Figure 14. This operator is the direct opposite of the *pick* operator and in fact it is produced by exchanging the *Prec* and *Post* lists of the latter.

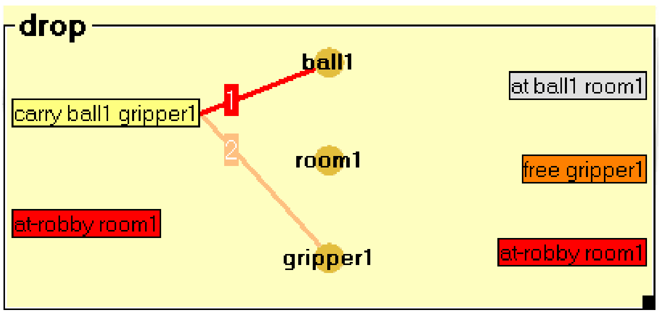


Figure 14. The drop operator

6.2. Problems

The designing of problems in the interface follows a similar model with that of operators. Problems can be formed by creating a list of objects, two lists of predicates and a number of connections among them. The list which is created by the predicates in the left column and the objects correspond to the initial state of the problem, while the goals are formed by the predicates of the right column.

Figure 15 presents a problem of the gripper domain, which contains two rooms (*rooma* and *roomb*), three balls (*ball1*, *ball2* και *ball3*) and the robot has two grippers (*left* and *right*). The initial state of the problem defines the starting locations of the robot and the balls and that both grippers are free. The goals specify the destinations of the three balls.

One of the key enhancements of ViTAPlan-2 over the past versions concerns the ability to use a predefined chain of objects that can be utilized for the definition of many kinds of facts. Consider for example the case where in gripper the moves of the robot are restricted by a relation (e.g. *connected*) which specifies which movements are feasible. If we suppose that the map of the rooms is the one presented in Figure 16, then this would require the user to add the *connected* relation 7 times in the problem's definition and each time to make the appropriate connections (see Figure 17). This can become a severe problem for large and complex maps.

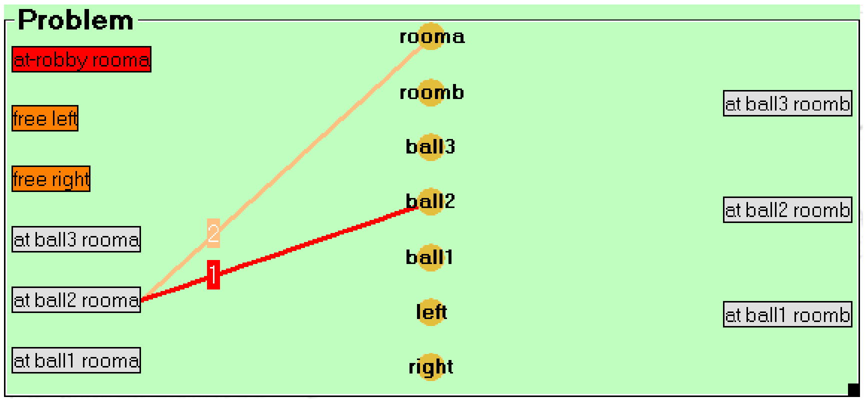


Figure 15. A Problem of the *gripper* domain

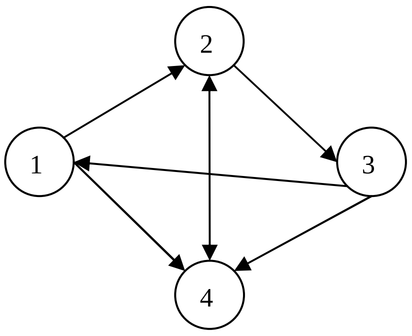

$$\begin{tikzcd}
 & 2 \arrow[dr] \arrow[dd, leftrightarrow] & \\
1 \arrow[ur] \arrow[dr] & & 3 \arrow[ll] \arrow[dl] \\
 & 4 &
\end{tikzcd}$$

Figure 16. A map for a *gripper* problem

There is a number of cases where similar maps are required. For example in the *hanoi* domain, according to the definition adopted by the planning community, the problem file must specify for each possible pair of pegs and discs which of the two is smaller. For a simple problem with three pegs and 5 discs, this yields to 39 smaller relations.

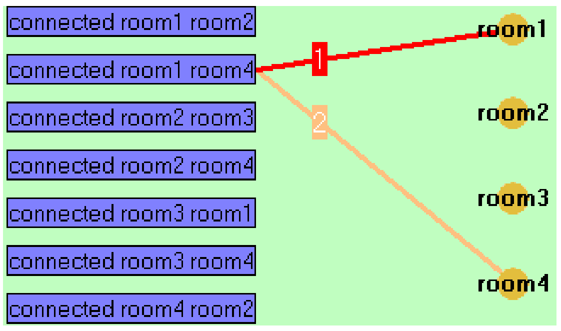


Figure 17. The connections between the rooms

In order to deal with this problem, ViTAPlan-2 contains a tool for building maps which makes it easier for the user to define multiple relations between pairs of objects that belong to the same entity. The user can use simple drag ‘n drop operations in order to define single and duplex connections between them. For example, the case described in Figure 17 can be easily encoded in ViTAPlan-2 using the map tool as shown in Figure 18.

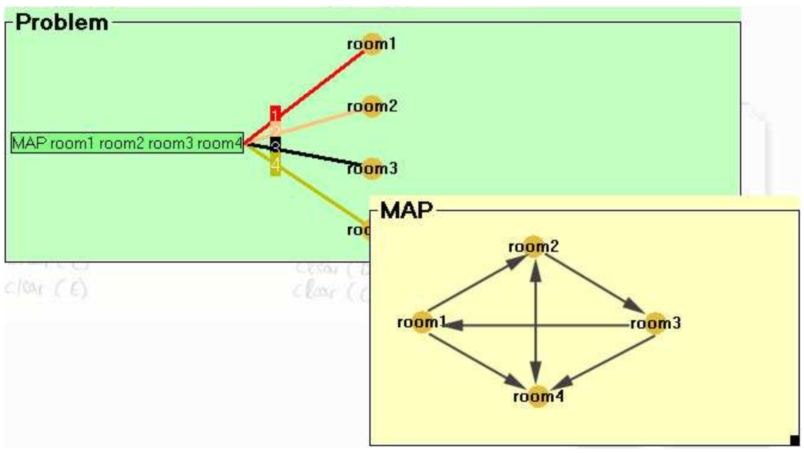


Figure 18. The map tool

6.3. Validity checking

The visual environment besides the automation and the convenient way in which new domains and problems are designed, it also performs a number of validity checks in the definitions, in order to assist the user I this really complex task. The entities-relations diagram, which is probably the most important part of the design of new domains is further used by ViTAPlan-2 as a reference model for the checks.

Each time the user tries to connect an instance of an entity *E* to an instance of a relation *R*, the tool checks if:

i. This specific connection generates a fact that has already been defined in the same operator or in the problem. In that case the connection is rejected, since PDDL does not allow redefinitions of the same fact inside the same scope.
ii. The definition of the relation in the entities-relations diagram contains a connection to the entity in which the *E* belongs.
iii. There is an empty slot in *R* which according to the diagram should be connected to objects of the same class as *E*’s.

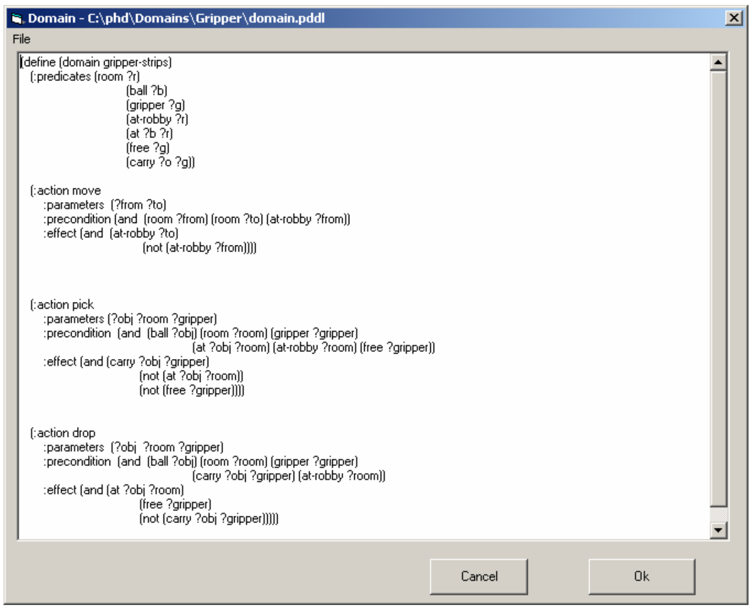
The checks listed above are performed dynamically also at each attempted change in the entities-relations diagram. For example, if the user deletes a connection between an entity and a relation, ViTAPlan will automatically delete any instance of this connection from all the operators and the definition of the problem. A similar update will also take place in the definition of the operators and in the problem, if the user deletes an entire relation or an entity.

Finally, a number of checks are also performed during the compilation of the design in PDDL. More specifically these checks contain:

- The case where no operator is defined
- The definition of empty operators
- The definition of void operators, i.e. operators with empty *Pre* and *Post* lists
- The definition of effect less operators (*Pre* ≡ *Post*)
- The definition of any empty problem
- Semi-connected edges which lead to incomplete facts

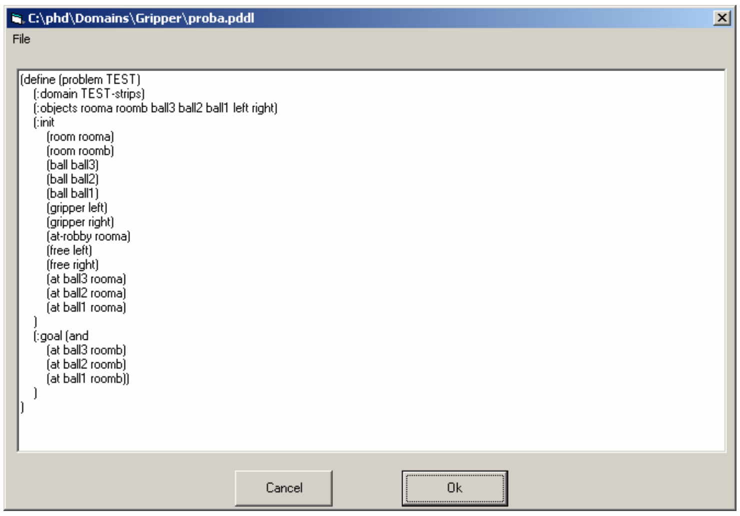
6.4. Translation to pddl

The domains and the problems that are designed in ViTAPlan are automatically compiled to the PDDL definition language, in order to allow their solving using different planning systems. The environment contains a PDDL editor which enables the user to see and even modify the PDDL files of his designs. For example, the gripper domain and the specific problem used in this section are presented in Figure 19 and Figure 20 respectively.



```
(define (domain gripper-strips)
  (:predicates (room ?r)
               (ball ?b)
               (gripper ?g)
               (at-robby ?r)
               (at ?b ?r)
               (free ?g)
               (carry ?o ?g))

  (:action move
     :parameters  (?from ?to)
     :precondition (and  (room ?from) (room ?to) (at-robby ?from))
     :effect (and  (at-robby ?to)
                   (not (at-robby ?from))))



  (:action pick
     :parameters (?obj ?room ?gripper)
     :precondition  (and  (ball ?obj) (room ?room) (gripper ?gripper)
                          (at ?obj ?room) (at-robby ?room) (free ?gripper))
     :effect (and (carry ?obj ?gripper)
                  (not (at ?obj ?room))
                  (not (free ?gripper))))


  (:action drop
     :parameters  (?obj  ?room ?gripper)
     :precondition  (and  (ball ?obj) (room ?room) (gripper ?gripper)
                          (carry ?obj ?gripper) (at-robby ?room))
     :effect (and (at ?obj ?room)
                  (free ?gripper)
                  (not (carry ?obj ?gripper)))))
```

Figure 19. The *gripper* domain in PDDL



```
(define (problem TEST)
   (:domain TEST-strips)
   (:objects rooma roomb ball3 ball2 ball1 left right)
   (:init
      (room rooma)
      (room roomb)
      (ball ball3)
      (ball ball2)
      (ball ball1)
      (gripper left)
      (gripper right)
      (at-robby rooma)
      (free left)
      (free right)
      (at ball3 rooma)
      (at ball2 rooma)
      (at ball1 rooma)
   )
   (:goal (and
      (at ball3 roomb)
      (at ball2 roomb)
      (at ball1 roomb))
   )
)
```

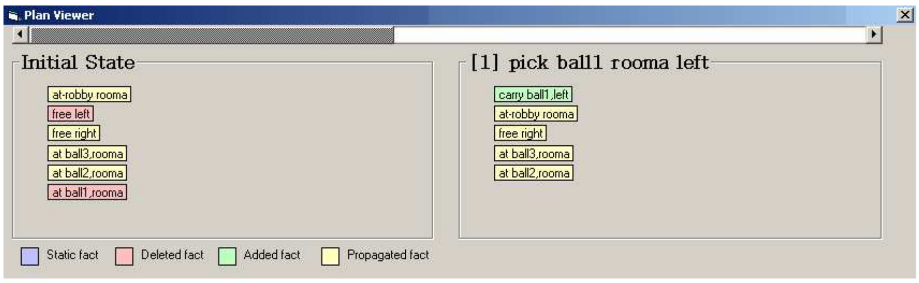
Figure 20. The PDDL file of a *gripper*'s problem

7. Plan Visualization

There are two modules for visualizing the plans in ViTAPlan: the execution simulation which simulates the execution of the plan in a virtual world and the steps visualization that presents the plan as an action graph.

7.1. Execution simulation

This module of ViTAPlan-2 allows the user to execute the plan that was acquired from the execution sub system and trace the changes that occur to the world as the actions of the plan are sequentially applied. Figure 21 presents the simulation module, which consists of three parts. The first one is a scroll bar, through which the user can browse for a specific action of the plan. The other two parts present the states of the world before the application of the selected action (left part) and after it (right part) respectively.



Initial State

```
at-robby rooma
free left
free right
at ball3,rooma
at ball2,rooma
at ball1,rooma
```

[1] pick ball1 rooma left

```
carry ball1,left
at-robby rooma
free right
at ball3,rooma
at ball2,rooma
```

🟦 Static fact 🟥 Deleted fact 🟩 Added fact 🟨 Propagated fact

Figure 21. Execution simulation

In the example presented in Figure 21 the first action of the plan is selected and therefore the state presented in the left part is the initial one, while the one in the right part is the first intermediate state. The tool uses proper color coding in order to discriminate the facts in four categories:

i. facts of the preceding state that are deleted by the action
ii. new facts that are added to the new state
iii. propagated facts
iv. static facts

7.2. Step graph

The second option for the user is to view the actions of the plan on a timeline, as shown in Figure 22. The timeline for the plan presents for each action the point in which it is scheduled to be executed and the facts in its precondition and add lists. Moreover, the visualization shows the interactions between the actions in the plan. More specifically, for each action *A* the user is able to see connections between:

i. each precondition of *A* and the most recent action that achieved it
ii. each fact that is deleted by *A* and the most recent action that established it
iii. each fact that is added by *A* and the following actions that have it in their preconditions

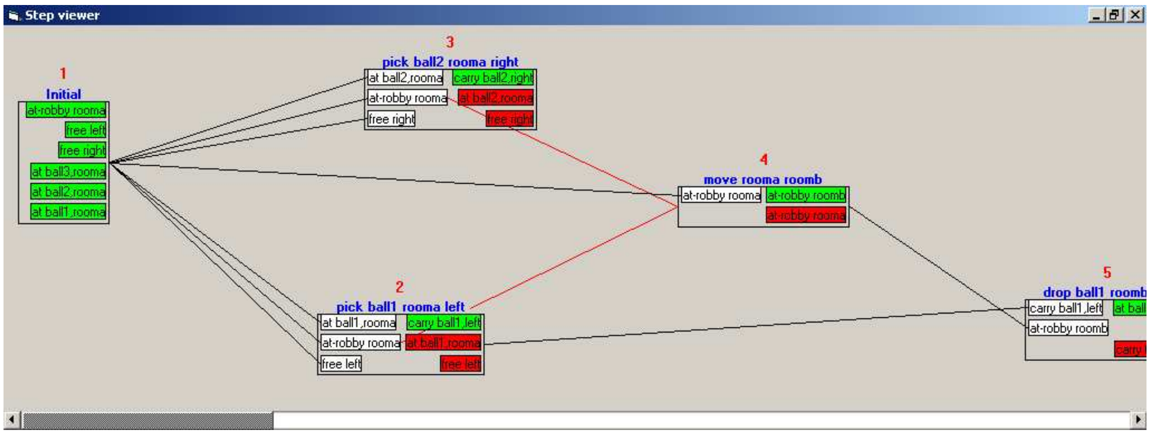


Figure 22. Step graph

The connections represented by the arcs in the graph present in graphical way all the interactions (positive and negative) between the steps of the plan and therefore allow the user to:

i. Comprehend the specific sorting in which the steps have been ordered and why a specific action should be placed before (or after) another one.

ii. Replace an action or a set of actions with other, taking care as not to violate the interactions, and thus produce modified or alternative plans.

iii. Find parallelizations in the plan execution and therefore reduce the total execution time and the cost of its application.

8. Conclusions and Future Work

This article presented ViTAPlan-2 a unified environment for automated planning which contains several modules with user friendly interfaces. One of the key features of the environment is the use of the configuration module that can automatically fine- tune the planning parameters based on rules, extracted from Machine Learning techniques, that associate planning parameters with problem attributes.

The current version of the environment has four main functions: a) using the planning system through a number of windows, controls and common dialogues, which makes it much easier for a non – programmer to use the planner and experiment with different setups of the planning parameters, b) use the Problem analyzer and the Rule system of HAP-RC in order to acquire useful knowledge about the morphology of each problem and automatically fine tune the planner with the most appropriate values for the planning parameters c) generating new domains and problems using a visual tool which saves the domain expert from the strict syntactic rules of PDDL, makes the definition of domains and problems more understandable, even for a non-planning-expert, and makes a number of consistency checks on the designs in order to generate PDDL files with as little flaws as possible and d) produce visual representations of the plans found by the planning system, which enable the user to better understand each step in the plan and also intervene and alter the plan at will.

In the future we plan to improve the interface in all functions of it and introduce others that will make it a complete tool for planning both for academic and industrial use. It is in our direct plans to enhance the tool for designing domains and problems with the ability to handle advanced aspects of the PDDL2.1[7], such as treatment of numerical values, explicit representation of time and duration, conditional effects e.t.c.

Acknowledgments

This research has been partially supported by SUN Microsystems, grant number: EDUD-7832-010326-GR.

References

1. A. Blum and M. Furst, Fast planning through planning graph analysis, In *Proceedings of the 14th International Conference on Artificial Intelligence*, (Berlin, Germany 1995), pp.636-642.
2. B. Bonet and H. Geffner, Planning as Heuristic Search. *Artificial Intelligence, Special issue on Heuristic Search*, 129/1-2 (2001), 5-33.

3. A. Botea, M. Enzenberger, M. Müller and J. Schaeffer, Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators, To appear in Journal of Artificial Intelligence Research (2005).
4. N. Bourbakis and A. Tascillo, An SPN-Neural Planning Methodology for Coordination of two Robotic Hands with Constrained Placement, Journal of Intelligent and Robotic Systems archive, 19-3 (1997), 321-337,
5. Y. Chen, B. W. Wah and C. Hsu, Subgoal Partitioning and Resolution for Temporal Planning in SGPlan, *Journal of Artificial Intelligence Research* (2005).
6. S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins and D. Tran, ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling, in *Proceedings of SpaceOps 2000*,(Toulouse, France, 2000).
7. M. Fox and D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20 (2003), 61-124.
8. A. Gerevini, A. Saetti and I. Serina, Planning through Stochastic Local Search and Temporal Action Graphs. *Journal of Artificial Intelligence Research,* 20 (2003), 239-290.
9. A. Gerevini, A. Saetti and I. Serina, Planning in PDDL2.2 Domains with LPG-TD, in *International Planning Competition, 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), abstract booklet of the competing planners*, (2004).
10. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld and D Wilkins, PDDL -- the planning domain definition language. *Technical report, Yale University*, New Haven, CT (1988).
11. J. Hoffmann and B. Nebel, The FF Planning System: Fast Plan Generation Through Heuristic Search, *Journal of Artificial Intelligence Research*, 14 (2001), 253-302.
12. J. Hoffmann, The Metric-FF Planning System: Translating ``Ignoring Delete Lists'' to Numeric State Variables, *Journal of Artificial Intelligence Research,* 20 (2003), 291-341.
13. Kautz, H. & Selman, B, Pushing the envelope: Planning, propositional logic and stochastic search. In Proceedings of the 13th National Conference on Artificial Intelligence, (Portland, Oregon 1996), pp. 1194-1201.
14. Kautz, H. & Selman, B., BLACKBOX: A new Approach to the application of theorem proving to problem solving, In Proceedings of the AIPS-98 Workshop on Planning as Combinatorial Search, (Pittsburgh, Pennsylvania 1998), pp. 58-60.
15. R. Kosara and S. Miksch, Metaphors of Movement: A Visualization and User Interface for Time-Oriented, Skeletal Plans. *Artificial Intelligence in Medicine,* Special Issue: Information Visualization in Medicine, 22 (2) (2001), 111-131.
16. K. Kundu, C. Sessions, M. DesJardins and P. Rheingans, Three-dimensional visualization of hierarchical task network plans, in *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, (Houston, Texas, 2002).
17. D. Long and M. Fox, Efficient Implementation of the Plan Graph in STAN, *Journal of Artificial Intelligence Research*, 10 (1998), 87-115.

18. T. L. McCluskey, D. Liu and R. Simpson, GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment, in Proceedings of the 13th International Conference on Automated Planning and Scheduling, (Trento, Italy,2003), pp. 92-101.
19. X. Nguyen, S. Kambhampati and R. Nigenda, AltAlt: Combining the advantages of Graphplan and Heuristics State Search, in *Proceedings of the 2000 International Conference on Knowledge-based Computer Systems*, (Bombay, India 2002).
20. J. Sanchez, A. D. Mali, S-MEP: A Planner for Numeric Goals, in Proceedings of the 5th IEEE International Conference on Tools with Artificial Intelligence (Sacramento, USA 2003), pp. 274-283.
21. G. Tsoumakas, D. Vrakas, N. Bassiliades & I. Vlahavas, Lazy Adaptive Multicriteria Planning, in *Proceedings of the 16th European Conference on Artificial Intelligence*, (Valencia, Spain 2004), pp. 693-697.
22. V. Vidal, A Lookahead Strategy for Heuristic Search Planning, in Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (Whistler, British Columbia, Canada 2004), pp. 3-7.
23. D. Vrakas and I. Vlahavas, Combining progression and regression in state-space heuristic planning. in *Proceedings of the 6th European Conference on Planning*, (Toledo Spain, 2002), pp. 1-12.
24. D. Vrakas, G. Gkioulekas, I. Refanidis, I. Sakellariou and I. Vlahavas, The PACOPLAN Project – A Parallel Constraint Planner with Java Interface. *http://lpis.csd.auth.gr/projects/pacoplan* (2002).
25. D. Vrakas and I. Vlahavas, A heuristic for planning based on action evaluation, in *Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems and Applications*, (Varna, Bulgaria, 2002), pp. 61-70.
26. D. Vrakas, G. Tsoumakas, N. Bassiliades and I. Vlahavas, Learning rules for Adaptive Planning, in *Proceedings of the 13th International Conference on Automated Planning and Scheduling*. (Trento, Italy, 2003), pp. 82-91.
27. D. Vrakas and I. Vlahavas, A Graphical Interface for Adaptive Planning, in *Proceedings of the Doctoral Consortium of the 13th International Conference on Automated Planning and Scheduling*, (Trento Italy, 2003), pp. 137-141.
28. D. Vrakas and I. Vlahavas, ViTAPlan: A Visual Tool for Adaptive Planning, in *Proceedings of the 9th Panhellenic Conference on Informatics*, (Thessaloniki, Greece, 2003), pp. 167-177.
29. D. E. Wilkins, T. J. Lee and P. Berry, Interactive Execution Monitoring of Agent Teams, *Journal of Artificial Intelligence Research*, 18 (2003), pp. 217-261.