WEB PLANNER: A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search

Maurício C. Magnaguagno, Ramon Fraga Pereira, Martin D. Móre and **Felipe Meneguzzi**
School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre - RS, Brazil
{mauricio.magnaguagno, ramon.pereira, martin.more}@acad.pucrs.br
felipe.meneguzzi@pucrs.br

Abstract

Automated planning tools are complex pieces of software that take declarative domain descriptions and generate plans for complex domains. New users often find it challenging to understand the plan generation process, while experienced users often find it difficult to track semantic errors and efficiency issues. To simplify this process, in this paper, we develop a cloud-based planning tool with code editing and state-space visualization capabilities. The code editor focuses on visualizing the domain, problem, and resulting sample plan, helping the user to see how such descriptions are connected without changing context. The visualization tool explores two alternative visualizations aimed at illustrating the operation of the planning process and how the domain dynamics evolve during plan execution.

1 Introduction

Classical planning algorithms typically require a declarative domain specification (often in PDDL (McDermott et al. 1998; Gerevini and Long 2005)) describing action schemata, which, in turn, define the dynamics of the underlying domain. Given the declarative nature of the formalism, planning algorithm implementations are often opaque regarding the intermediate steps between reading the formalism and generating a plan. Thus, writing such specifications may be a challenging task for new users even for simple domains, while detecting semantic mistakes in complex domains is always non-trivial. Practical applications of classical planners require not only a formalization of the domain in PDDL that is correct, but also takes advantage of the search mechanisms employed by the underlying planners to find solutions efficiently.

Most modern classical planning solvers (Hoffmann and Nebel 2001; Helmert 2006; Richter and Westphal 2010; Hoffmann 2011) use heuristic functions to estimate which states are likely to be closer to the goal state and save time and memory during the planning process. Different planning domains may require different heuristic functions to focus the search on promising branches and be solved within a reasonable time with little memory footprint. Thus, key to understanding the efficiency of a domain formalization is its impact on the heuristic function used by the underlying planner.

Even when the user successfully compiles and executes a planning instance with the chosen heuristic function the planner may fail to find a correct plan for the intended domain. In these cases, virtually no planning algorithm offers extra information, and the user only knows that the domain or problem are described in a way that makes it impossible to find a valid plan. Finally, since most planners are academic projects made to execute under very specific environments they lack a clear documentation to guide new users in the compilation process, while a web-based planner offers planning algorithms with no setup time.

This paper describes a tool to address the challenges of helping a domain expert to tune a formalization to any planning heuristics and spotting semantic errors in planning domains. Our tool, which we describe in Section 3, includes a PDDL code editor with syntax highlight and auto-complete aimed at helping users to efficiently develop PDDL domains in a similar workflow to many popular integrated development environments (IDEs). Importantly we integrate the editor to two visualization tools, described in Section 2, developed to help users cope with the declarative nature of PDDL and explore the effects of changes to the domain in solving concrete problems. First, we use a visual metaphor from the literature to see how a plan execution achieves (or does not) a goal state from an initial state (Magnaguagno, Pereira, and Meneguzzi 2016). Second, we develop a new state-space search visualization that uses tree drawing (in both cartesian and radial layouts) in conjunction with heatmaps to represent how the distance (*e.g.*, how colder or warmer) to the goal state changes during search. We use a case study in Section 4 to illustrate how our approach works and validate our approach from user tests, which we describe in Section 5 showing the results we obtained from employing the tool in a planning course. In Section 6, we survey related work on planning tools and data visualization, and conclude the paper in Section 7 discussing our conclusions and future work.

2 Background

2.1 Planning

Planning is the problem of finding a sequence of actions (*i.e.*, plan) that achieves a particular goal from an initial state (Ghallab, Nau, and Traverso 2004). A state is a finite set of facts that represent logical values according to some

interpretation. Facts are divided into two types: positive and negated facts. Predicates are denoted by an n-ary predicate symbol applied to a sequence of zero or more terms. An operator is represented by: a name that represents the description or signature of an action; a set of preconditions, *i.e.*, a set of facts or predicates that must be true in the current state to be executed; a set of effects, which has an add-list of positive facts or predicates, and a delete-list of negative facts or predicates. An action is an instantiated operator over free variables. A planning instance is represented by: a domain definition, which consists of a finite set of facts and a finite set of actions; and a problem definition, which consists of an initial state and a goal state. The solution of a planning problem is a plan, which is a sequence of actions that modifies the initial state into one in which the goal state holds by the successive execution of actions in a plan. To formalize planning instances, we use the STRIPS (Fikes and Nilsson 1971) fragment of PDDL (McDermott et al. 1998), which contains domain and problem definition in different files.

Heuristic functions are used to estimate the cost of achieving a particular goal (Ghallab, Nau, and Traverso 2004). In classical planning, this estimate is often the number of actions to achieve the goal state from a particular state by exploring only promising states. Estimating the number of actions is a NP-hard problem (Bylander 1994). In automated planning, heuristics can be domain-dependent or domain-independent, and a well-tuned heuristic can result in a substantial reduction in search time by pruning a vast part of the state-space.

2.2 Data Visualization

Visualization techniques aim to convey some kind of information using graphical representation (Ward, Grinstein, and Keim 2015). The use of data visualization techniques is often associated to a set of data with the aim of communicating a particular information clearly and efficiently via graphical representation.

Data visualization techniques are concerned with what is the best way to display a dataset, for instance, how to display relation information. Relation information can be displayed efficiently by using hierarchies that convey relation information. Edges in a hierarchical tree represent a relation between nodes. A Cartesian tree visualization is a way to display hierarchical trees as a coordinate system. A radial tree visualization is a way to display a hierarchical tree structure in which such tree expands outwards and radially. In Subsection 3.2 we explore such tree visualizations. Besides hierarchical visualization, we highlight other visualization methods that are closely related to the ones we develop in this work, such as *Gantt charts* (Wilson 2003), which are used to show how tasks are correlated and how much time is expected to complete them, *Waveforms* (Ha 2010, Chapter 1 – page 2) are used to express the behavior of analog or digital data through time, and Heatmap visualization (Ward, Grinstein, and Keim 2015), which uses a color scheme to illustrate values in a graphic in which each color in the scheme represents one limit value and the many values in the interval are represented by the mix of such colors.

3 WEB PLANNER Architecture

We designed our tool envisioning a development process centered around two tasks by the domain developer. In the first task, the user aims to describe both domain and problem correctly. In the second task, the user tries to identify details of the description (in terms of predicate use) that impact performance and how these predicates appear during the planning process. The domain designer is free to move between these tasks and repeat until satisfied with the results. Once a planning instance is described it is possible to visualize the explored state-space, even when the planning process fails. When the planning process returns a plan the user is able to visualize how predicates were added or deleted by each action in the plan. Such interface could also help planning system developers to explore how planners in development behave.

To avoid the considerable setup time of some planner implementations and maintain a consistent interface across platforms, we use a web interface. The planner is executed in the server, while the editor, output and visualizations are displayed and executed in the browser. The communication between the two sides uses JSON[1]. Figure 1 shows the architecture of the WEB PLANNER.

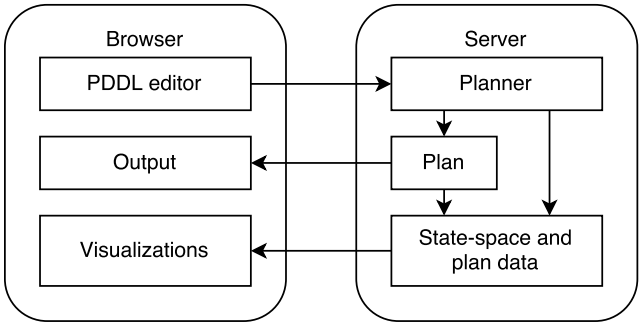

Figure 1: Overview of the WEB PLANNER Architecture.

3.1 Domain Development Interface

To better describe planning domains and problems, we identified some requirements to improve the process of editing such descriptions, as follows:

- PDDL syntax highlight and auto-complete to alleviate user learning curve. For example, to define a new action, our PDDL editor provides an action-template (an auto-complete function of our editor, pressing *CTRL+Space* after typing the word *action*) that shows how an action is defined in PDDL, as shown in Figure 2. Our editor also provides templates for domain and problem description, just pressing *CTRL+Space* after typing the word *domain* or *problem*, respectively;
- See and edit both domain and problem simultaneously, avoid going back and forth between descriptions gives a

[1]JSON (JavaScript Object Notation) is an open-standard format for structuring data.

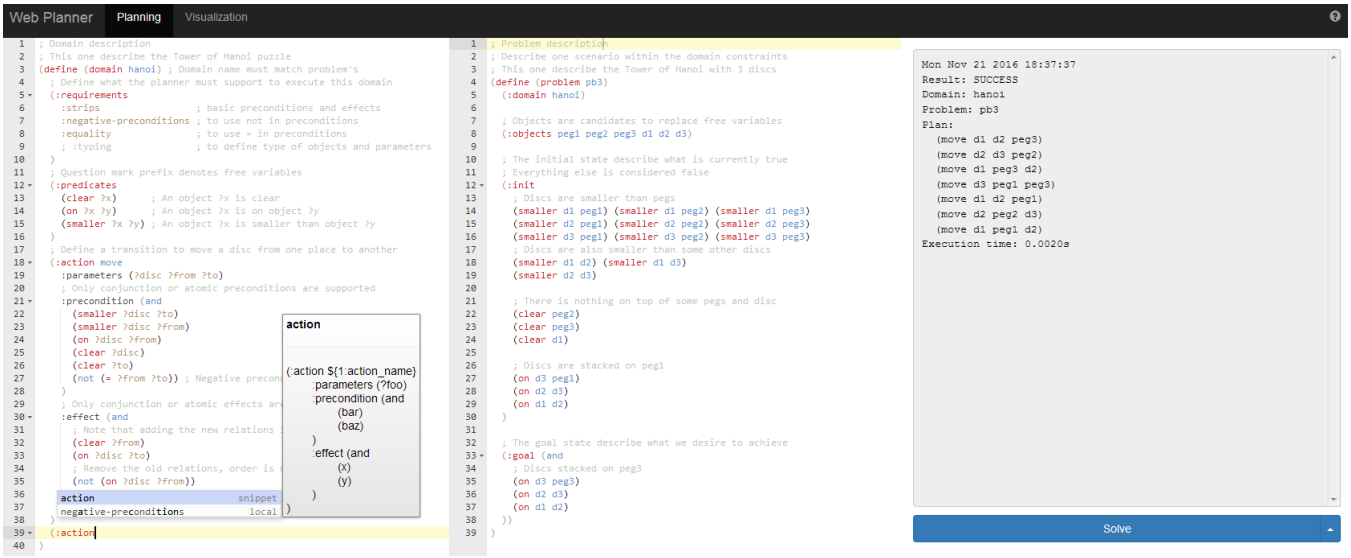

Figure 2: WEB PLANNER editor interface with domain editor (left), problem editor (center) and plan output (right). Action template is provided by autocomplete.

better idea of them being used together while minimizing the user effort; and

- Execute the current planning instance without a context change;

To meet such requirements, we split the editor interface horizontally in 3 parts: domain, problem, and planner output. The ability to see input alongside output is very important for both advanced users, that are modifying or extending legacy PDDL, and new users, such as students, that are not used with the domain and problem distinction. Instead of starting with a blank planning instance we opted for a simple but complete Towers of Hanoi example to be loaded by default.

The solve button sends the planning instance to the server to obtain an output based on the domain and problem descriptions contained in the editor. Our editor uses brace, a variant of the ace editor, and it is able to highlight most PDDL elements, some of which are currently not supported by the back-end planner. The output provided by the planner contains the plan and execution time when successful, error messages when the parser fails, or a failure message when no plan is found. Due to screen space limitations and demand, the visualizations were left to a secondary interface, as users can only visualize after the initial description step.

3.2 Visualization Interface

We currently support two visualizations, one focusing on the explored state-space and the other on the execution of the first plan found.The impact of heuristics in the state-space is often introduced in AI lectures using images, such as the ones from Figure 3, to show how the contour of the explored states grows in all directions on blind search and towards the goal state in informed search (using heuristics) (Russell and Norvig 2009, Chapter 3 – page 97). Such images target an audience new to the concept of using a computed auxiliary function to speed-up search. More interesting examples are possible with animations on a grid, showing the step-by-step process of search. Since not all domains can be mapped to a grid, the visualization process is often limited to path-finding domains. To generate such contours we opted for a tree-based visualization.

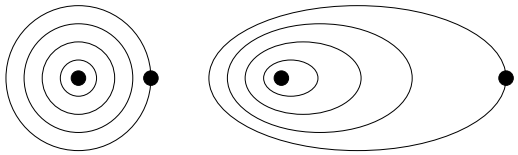

Figure 3: Search contours are defined by search mechanism and heuristic function, either equally exploring in all directions (left) or giving priority towards the goal state (right).

Heuristic Visualization: The heuristic visualization we developed takes advantage of interactive elements to avoid information overload while providing alternative layouts, cartesian and radial tree visualizations. The radial layout matches the abstraction used by heuristic examples while the cartesian layout generates a visualization more compact. In practice, we use the Reingold-Tilford algorithm[2] to display both tree layouts. Using tree visualizations we aim to show how planning heuristics explore the state-space to achieve a particular goal.

To compare and explore the state-space of a planning instance, we implemented two planning methods. The first method is based on breadth-first search, and thus uses no

[2]Reingold-Tilford is an algorithm for an efficient tidy arrangement of layered nodes. We use an implementation based on a D3 example available at: `http://bl.ocks.org/mbostock/4063550`.

heuristic, exploring the state space in the order of distance from the initial state. The second method implements greedy best-first search using Hamming distance (Hamming 1950) as a heuristic. Our visualization tool supports other search mechanisms and heuristic functions as long as such mechanisms search through the state-space, the selected ones are used only as examples.

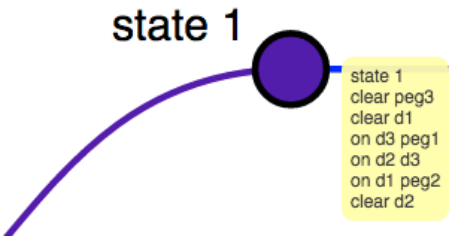

Figure 4: Tooltip that displays the set of instantiated predicates in a state. This figure illustrates state 1 and its predicates for a planning instance of the Hanoi domain.

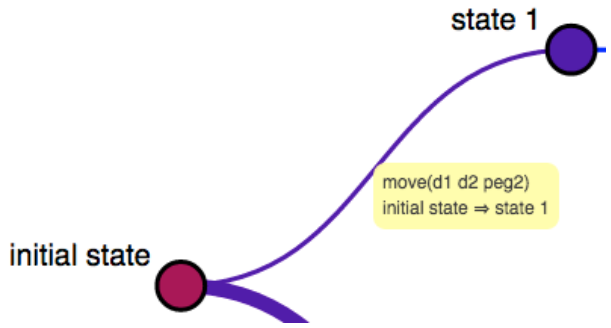

Figure 5: Tooltip that displays the instantiated action applied between two states. This figure illustrates state 1 and its predicates for a planning instance of the Hanoi domain.

With the explored state-space and heuristic information about each state we use an hierarchical tree to represent the data obtained from the planning process. In this tree, each node represents a state (*i.e.*, a set of instantiated predicates), and an edge represents a state-transition (*i.e.*, the execution of an action). The root node represents the initial state. Our visualization displays the state-space of a planning heuristic by coloring the estimated distance between states using a heatmap. Information such as the content of the state or the applied action is hidden until the user hovers that position with the cursor to display such data in a tooltip, as show in Figures 4 and 5. Nodes and edges are colored according to the estimated distance to the goal state. Red nodes represent states closer to the goal state, *i.e.*, warmer, while distant nodes are represent by blue, *i.e.* colder. The heuristic gradient is defined in Figure 6. It is important to highlight that the estimated distance can be different according to the used heuristic. Therefore, the state-space search can reach different states in a different order according to the heuristic used, for example, Figures 10 and 11. As more states are explored the more visible the contours are, as seen in Figure 7. If planning is successful the edges from initial to goal node are emphasized. The trees are also generated to failed planning instances, which can be used for debug purposes.

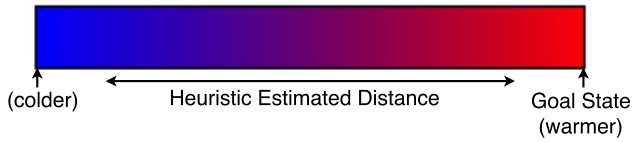

Figure 6: Color scheme that our visualization tool uses to represent the estimated distance.

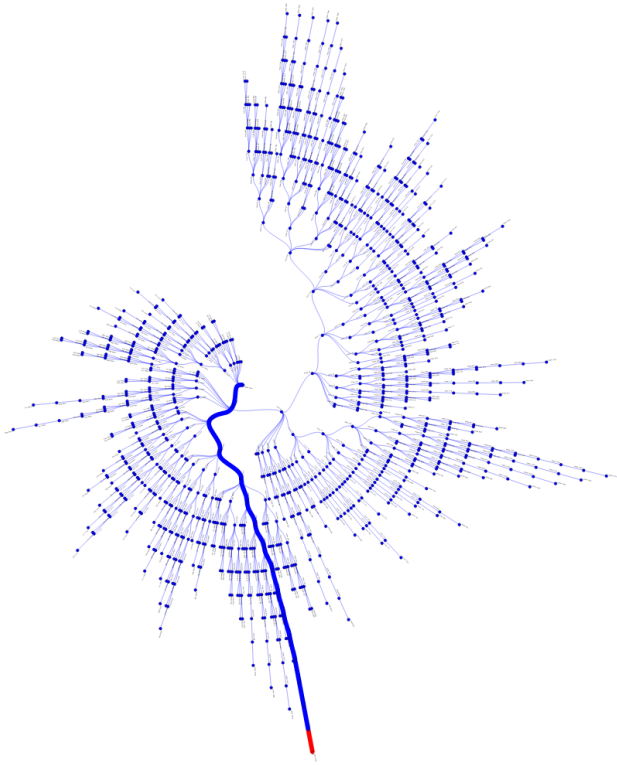

Figure 7: Contours become visible as more states are explored. This planning instance obtain all goal predicates at the same time, which makes the heatmap mostly blue (colder), while the goal state is located at the bottom in red (warmer).

Dovetail Metaphor Visualization: The second visualization we implemented is a visual metaphor called Dovetail (Magnaguagno, Pereira, and Meneguzzi 2016), which is useful to see how predicates change along the plan execution. Each ground predicate that appear in an action effect is represented as one line while both initial state, goal state and actions are represented as columns. Our interface allows a user to move and zoom to parts of this visualization (illustrated in Figure 8), with tooltips providing extra information as shown in Figure 9 for the domain of the case study of Section 4. The use of this visual abstraction (Dovetail) aims

to improve the learning curve for defining and debugging planning domains and problems.

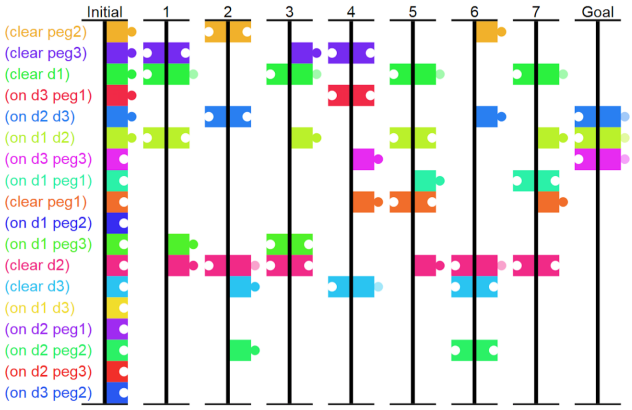

Figure 8: Dovetail plan visualization of Hanoi domain with 3 discs and a plan of size 7.

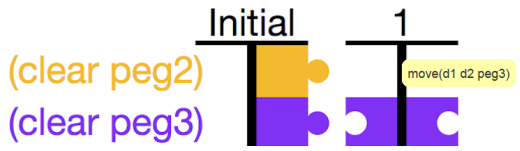

Figure 9: Tooltip that displays the instantiated action in a plan on Dovetail.

4 Case Study

To validate our tool, we developed a case study of a planning instance using different planning heuristics displaying the state-space.We selected the Hanoi towers domain to illustrate what can be expected from the visualizations. The Hanoi domain describes the towers of Hanoi problem, where one must move a stack of discs from one peg to another without stacking a larger disc onto a smaller one, three pegs are available in total. Problem instances of this domain show that the goal cannot be accomplished in an incremental way, requiring the plan to build and destroy partial towers several times to obtain the complete tower in the final peg. Domains that present such behavior are not pruned as much as others by the Hamming distance as a heuristic function and have a visible color fluctuation between the gradient limits instead of a clear movement towards red, as seen in the Cartesian tree of Figure 10. The Cartesian tree is better to represent and compare such smaller graphs, while the radial tree highlights the side to which the heuristic gave priority during search, as seen in Figure 11, where the top-left branch was not explored. Other domains may suddenly reach a goal state from a mostly blue colored graph, in which all states are far away from the goal, as seen in Figure 7, or incrementally reaching the goal clearly going from one extreme of the gradient to the other, as in the Logistics domain.

To better understand how the predicates are affected by the plan we use the Dovetail metaphor. This particular Hanoi

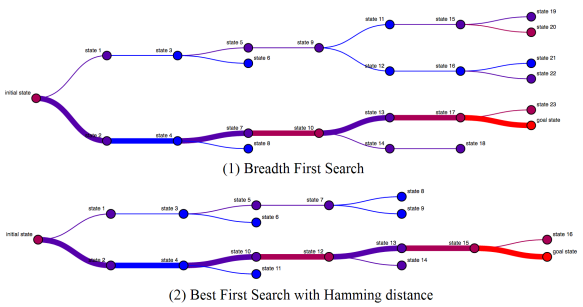

Figure 10: Cartesian tree visualizations of the state-space of Hanoi with 3 discs.

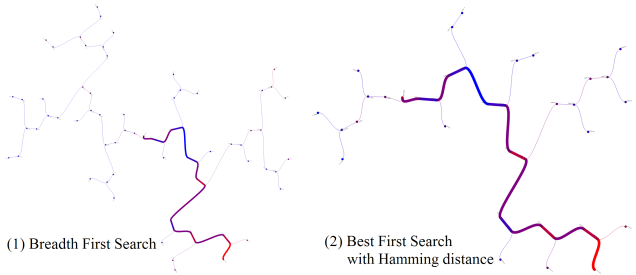

Figure 11: Radial tree visualizations of the state-space of Hanoi with 3 discs.

planning instance is solved by a 7-step plan, represented by the pieces labeled with numbers at the top, Figure 8. Each piece has preconditions represented on the left side and effects represented on the right side. In this case we can see the first action, *move(d1 d2 peg3)*, moving a clear disc *d1* that starts on disc *d2* to a clear peg *peg3*, leaving *d2* clear and *peg3* not clear. We can see the predicate *clear d1* being tested by each odd-index action, revealing the pattern of movements related with the disc *d1*.

5 Survey Results

To evaluate WEB PLANNER, a group of four users from our automated planning course[3] were asked to fill a survey after using the tool to describe the *RPG* domain from the International Competition on Knowledge Engineering for Planning and Scheduling [4]. The survey contained the following questions and answers:

- How familiar are you with automated planning languages and algorithms?
  - Only 2 users have used PDDL before.
- Did Web-planner visualizations help you to find any bugs/errors/interesting points during the course of your task?
  - One user found missing preconditions.
- Mark other planners/tools you used in your experiments:

[3] http://github.com/pucrs-automated-planning/syllabus
[4] http://ickeps2016.wordpress.com

- Fast-Downard (1), JavaFF (1), JavaGP (3), Planning.domains (3), STRIPS-Fiddle (1)

- Which features you missed the most?
  - Support more requirements (2), Auto-complete (1), Option to clear console (1), Find (common) errors in PDDL (1).

Results of system reaction show evidence of the utility of our tool, albeit with many suggested improvements, in Figure 12 with minimum, maximum and average represented. The current planning output must be improved in order to provide more meaningful messages about errors while taking advantage of the integrated editor to draw attention to specific lines where parsing errors were detected. Other improvements are more related to the editor itself, making it more flexible to attend different user needs, such as theme, font size and the ability to re-size each part of the editor. Users also asked for more planners/requirements to be supported.

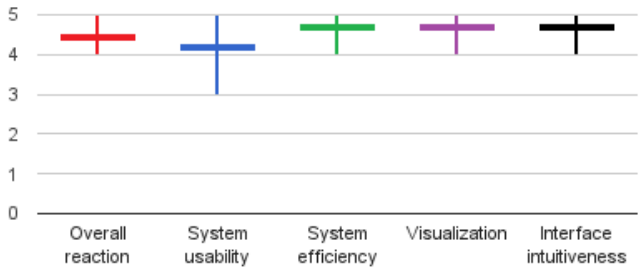

Figure 12: Survey results, users were asked to evaluate the system between frustrating (0) and satisfying (5).

6 Related work

We now discuss related work and tools that are used to formalize planning domains, visualize changes on a large amount of hierarchical data, and visualize state-space search algorithms.

`Planning.Domains`[5] is a collection of web tools for automated planning. These web tools provide a web PDDL editor, an API that contains a wide collection of PDDL benchmark domain and problem files (most of them used on the International Planning Competition), and a planner in the cloud that allows using not only a planning solver, but also VAL (Howey, Long, and Fox 2004), a plan validation tool. Similar to our approach, `Planning.Domains` provides a PDDL editor, however, our approach provides not only a web editor with syntax highlighting, but also a set of tools to develop and visualize planning domains using metaphors and alternative data visualization methods.

To edit PDDL domains and problems, we highlight two approaches, `myPDDL` and PDDL Studio. `myPDDL`[6] (Strobel and Kirsch 2014) is an editor extension for Sublime Text, which provides PDDL syntax highlighting, snippets, and domain visualization (*e.g.*, diagram types). PDDL Studio (Plch et al. 2012) is an IDE to edit PDDL domains and problems. This IDE provides syntax highlighting, code completion, and context hints specifically designed for PDDL.

Graphical Interface for Planning with Objects (GIPO) (Simpson, Kitchin, and McCluskey 2007) is a tool for planning domain knowledge engineering that allows the specification of domains in PDDL and *Hierarchical Task Network* (HTN). Besides domain knowledge engineering, GIPO provides an animator tool to graphically inspect the plans produced by the internal planner, given a domain and problem specification. Unlike our approach, GIPO checks a set of plans to validate a domain and problem specification, indicating whether the domain and problem specification do support the given plans. Similar to Dovetail metaphor we implemented in WEB PLANNER, GIPO also provides an animator tool to visualize how a sequence of actions (*i.e.*, a plan) connects to form a plan that achieves a goal state from an initial state. *VisPlan* (Glinskỳ and Barták 2011) is an interactive tool to visualize and verify plans' correctness. This tool is closely related to Dovetail metaphor in the sense of helping planning users to better understand how a sequence of actions achieve a goal from an initial state. *VisPlan* identifies possible flaws (*i.e.*, incorrect actions) in a plan, allowing users to manually modify this plan by repairing these identified flawed actions.

PDVer (Raimondi, Pecheur, and Brat 2009) is a methodology and tool that verifies if a PDDL domain satisfies a set of requirements (*i.e.*, planning goals). This tool allows an automatic generation of these requirements from a *Linear Temporal Logic* (LTL) specification into a PDDL description. This tool is concerned with how the corresponding PDDL action constraints are translated from an LTL specification. Whereas *PDVer* provides a summary of test cases (positive and negative) indicating why a PDDL domain specification does not satisfy a set of requirements to achieve a goal.

itSimple (Vaquero et al. 2012) is concerned with domain modeling, using steps to guide the user from informal requirements (UML) to an objective representation (Petri Nets). The itSimple features provide a visualization and simulation tool to help understanding planning domains through diagrams. *itSimple* uses UML diagrams to model planning instances and Petri Nets for validating planning instances.

Magnaguagno *et al.* (2016) developed a visual metaphor to visualize and learn how the planning process works. We have applied this visual metaphor in our web tool by using colors for different instantiated predicates in a state along a plan execution. Dovetail results suggest that this visual metaphor can be useful to define and debug the planning process.

We found two approaches to data visualization suitable for heuristics. In (Kuwata and Cohen 1993), Kuwata and Cohen develop visualization methods to understand and analyze the search-space and behavior of heuristic functions, by exploring the usefulness of these methods on shaping state-space search. The heuristic functions they explore are A* and IDA*. Tu and Shen (2007) propose a set of strategies to visualize and compare changes in hierarchical data using treemaps.

[5] `http://planning.domains`

[6] `http://github.com/Pold87/myPDDL`

7 Conclusions

In this paper, we report on a cloud-based planning tool we developed, which consists of a PDDL editor to formalize planning domains and problems, and visualizations to help understand the effect of planning heuristics in the domains. This work aims to simplify the setup process required to execute planners while providing visualizations to better understand how domain differences and heuristics can impact the performance of the planner. A small-scale survey indicates promising results while asking for improvements that are already in development.

As future work, we intend to support user-defined heuristics in our planner along with alternative options to the user, such as selectable color schemes for the visualization and a side-by-side state-space view for comparison. We believe that such tool can help new heuristics to be developed and tested, giving the user a better grasp of the impact of heuristics to the state-space exploration, which is usually an invisible entity. Instead of outputting only a plan and the time to compute it in the editor interface, we expect to add not only better parsing error messages but also detection of bad constructions, such as unnecessary requirements or effects that are equal to preconditions. Our WEB PLANNER tool is available at `http://web-planner.herokuapp.com`.

Acknowledgments

We acknowledge the support given by CAPES/Pro-Alertas (88887.115590/2015-01) and CNPQ within process number 305969/2016-1 under the PQ fellowship.

References

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Journal of Artificial Intelligence Research (JAIR)* 69:165–204.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Journal of Artificial Intelligence Research (JAIR)* 2(3):189–208.

Gerevini, A., and Long, D. 2005. Plan Constraints and Preferences in PDDL3. *The Language of the Fifth International Planning Competition. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning - Theory and Practice.* Elsevier.

Glinskỳ, R., and Barták, R. 2011. Visplan–interactive visualisation and verification of plans. *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)* 134–138.

Ha, T. T. 2010. *Theory and design of digital communication systems*. Cambridge University Press.

Hamming, R. W. 1950. Error detecting and error correcting codes. *Bell System Technical Journal* 29(2):147–160.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)* 14(1):253–302.

Hoffmann, J. 2011. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Computing Research Repository (CoRR)* abs/1106.5271.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*, 294–301.

Kuwata, Y., and Cohen, P. R. 1993. Visualization Tools for Real-Time Search Algorithms. *Computer Science Technical Report*.

Magnaguagno, M. C.; Pereira, R. F.; and Meneguzzi, F. 2016. DOVETAIL - An Abstraction for Classical Planning Using a Visual Metaphor. In *Proceedings of FLAIRS, 2016.*

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL − The Planning Domain Definition Language. *Technical Report – Yale Center for Computational Vision and Control*.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL Studio. In *Proceedings of ICAPS'09*, 15–18.

Raimondi, F.; Pecheur, C.; and Brat, G. 2009. PDVer, a Tool to Verify PDDL Planning Domains. In *Proceedings of ICAPS'09 Workshop on Verification and Validation of Planning and Scheduling Systems, Thessaloniki, Greece*.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39(1):127–177.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd edition.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Eng. Review* 22(2):117–134.

Strobel, V., and Kirsch, A. 2014. Planning in the Wild: Modeling Tools for PDDL. In *Joint German/Austrian Conference on Artificial Intelligence*, 273–284. Springer.

Tu, Y., and Shen, H. W. 2007. Visualizing Changes of Hierarchical Data using Treemaps. *IEEE Transactions on Visualization and Computer Graphics* 13(6):1286–1293.

Vaquero, T.; Tonaco, R.; Costa, G.; Tonidandel, F.; Silva, J. R.; and Beck, J. C. 2012. itSIMPLE 4.0: Enhancing the modeling experience of planning problems. In *Proceedings of ICAPS'12*, 11–14.

Ward, M. O.; Grinstein, G.; and Keim, D. 2015. *Interactive Data Visualization: Foundations, Techniques, and Applications, Second Edition - 360 Degree Business*. Natick, MA, USA: A. K. Peters, Ltd., 2nd edition.

Wilson, J. M. 2003. Gantt charts: A centenary appreciation. *European Journal of Operational Research* 149(2):430 – 437.