Chapter 11
Web Planner: A Tool to Develop, Visualize, and Test Classical Planning Domains

Maurício C. Magnaguagno , Ramon Fraga Pereira , Martin D. Móre , and Felipe Meneguzzi

Abstract Automated planning tools are complex pieces of software that take declarative domain descriptions and generate plans from domains and problems. 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. In response, we develop a cloud-based planning tool with code editing and state-space visualization capabilities that simplifies this process. The code editor focuses on visualizing the domain, problem, and resulting sample plan, helping the user 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.

Keywords Classical planning · STRIPS · PDDL · State-space visualization

1 Introduction

Classical planning algorithms typically require a declarative domain specification describing action schemata, which, in turn, define the dynamics of the underlying domain. Since the inception of the International Planning Competition (IPC) [24], the standard specification language for classical planning is the Planning Domain Definition Language (PDDL) [3, 15]. Given the declarative nature of PDDL, planning algorithm implementations are often opaque regarding the intermediate steps between reading the formalism and generating a plan. This creates a twofold problem for domain engineers that wish to use automated planning to solve

M. C. Magnaguagno (✉) · R. F. Pereira · M. D. Móre · F. Meneguzzi
Pontifical Catholic University of Rio Grande do Sul (PUCRS), School of Technology, Porto Alegre, RS, Brazil
e-mail: mauricio.magnaguagno@acad.pucrs.br; ramon.pereira@acad.pucrs.br; martin.more@acad.pucrs.br; felipe.meneguzzi@pucrs.br

M. Vallati, D. Kitchin (eds.), *Knowledge Engineering Tools and Techniques for AI Planning*, https://doi.org/10.1007/978-3-030-38561-3_11

problems: ensuring the correctness of each domain description, and optimizing the efficiency of a planning algorithm for each domain description.

First, regarding correctness, writing PDDL specifications may be a challenging task for new users even for simple domains, while detecting semantic mistakes in complex domains is always non-trivial. 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 either the domain has some kind of description error or that specific problem supplied to the planner is unsolvable, such that the planner cannot find a correct plan.

Second, practical applications of classical planners require not only a formalization of the domain in PDDL that is correct, but also exploit the search mechanisms employed by the underlying planners to find solutions efficiently. Most modern classical planning solvers [8, 9, 11, 19] 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.

In order to address these challenges, we developed WEB PLANNER, an online tool aimed at helping domain engineers to tune a formalization to a number of common planning heuristics and spotting semantic errors in planning domains. Our tool, which we describe in Sect. 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 Sect. 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 [14]. 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 conducted a structured case study (described in Sect. 4.1) to illustrate how our approach works and validate from user tests, which we describe in Sect. 4.2 showing the results we obtained from employing the tool in a planning course. WEB PLANNER has been deployed for 2.5 years as openly available tool for the planning community, which allowed us to collect anonymous usage statistics. In Sect. 5, we survey related work on planning tools and data visualization, and conclude the paper in Sect. 6 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 [4]. 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 [2] fragment of PDDL [15], which contains domain and problem definition in different files.

Heuristic functions are used to estimate the cost of achieving a particular goal [4]. 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 [1]. 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 [26]. 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 Sect. 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* [27],

which are used to show how tasks are correlated and how much time is expected to complete them, *Waveforms* [6, Chapter 1—page 2] are used to express the behavior of analog or digital data through time, and heatmap visualization [26], 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 its 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 a server, while the editor, output and visualizations are displayed and executed in the browser. The communication between the two sides uses JSON.[1] Figure 11.1 shows the architecture of WEB PLANNER.

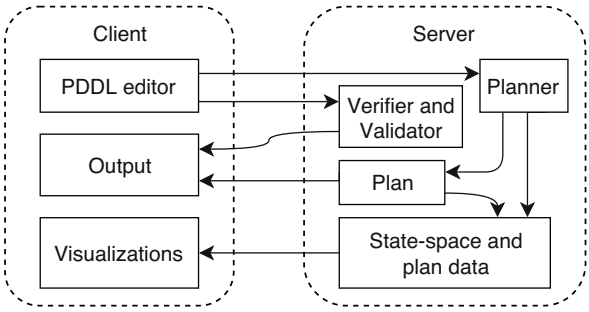

Fig. 11.1 Overview of WEB PLANNER architecture

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

3.1 Domain Development Interface

To better describe planning domains and problems, we identified three key requirements to improve editing such descriptions. First, we required our tool to provide the two common IDE features of syntax highlighting, code auto-completion, and templates (PDDL snippets) to streamline the editing process. 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 illustrated in Fig. 11.2. Besides templates for PDDL actions, our editor also provides templates for domain and problem description, just pressing *CTRL+Space* after typing the word *domain* or *problem*, respectively. Second, we the interface must show both domain and problem simultaneously, to avoid forcing the user to go back and forth between descriptions or browser tabs. This interface arrangement improves the designer's awareness of the interactions between a domain and instances of its problems and minimizes user effort in terms of required interface actions (i.e., key presses and mouse clicks). Finally, our interface must include a visualization and an action button in the same context as the editors, allowing the designer to execute the current planning instance without a changing context.

To meet such requirements, we split the editor interface horizontally in three parts: domain, problem, and 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.

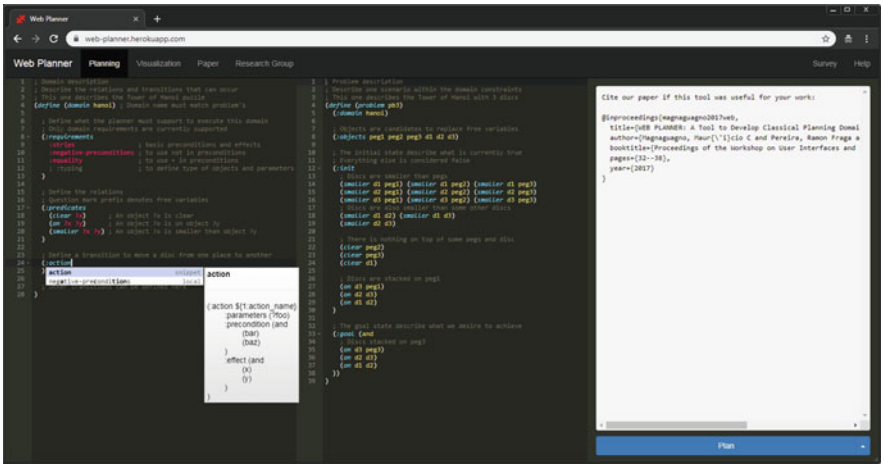

Fig. 11.2 WEB PLANNER editor interface with domain editor (left), problem editor (center), and plan output (right). Action template is provided by auto-complete shortcuts. Verification and validation tools available through caret button

The **Plan** 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*,[2] a variant of the *ace editor* without server-side processing to highlight programming language elements. In our case most PDDL elements are highlighted, 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, the visualizations were left to a secondary interface, as users can only visualize after an initial description step. Our goal is to make clear that domain and problem are described together, while planning insights and optimization steps can be obtained later, if required, without overloading the user with information.

Verifier and Validator Plan output alone is not enough to identify errors in a planning description. The declarative nature of PDDL obscures the intermediate structures of the planner for novice users (or users without working knowledge of planner implementation), requiring further modification of the chosen planner to log such information. To address this problem we provide two extra tools to find description errors and mistakes in their domain and problem. The first is a *verifier*, a tool that finds common mistakes in both domain and problem descriptions. The second is a *validator*, a tool that tries to execute a plan provided by the user in the domain and problem previously described, and reports any errors found while doing so. Tests cover only atomic or conjunctive preconditions and effects, limited to *:strips*, *:negative-preconditions*, *:equality*, and *:typing* requirements. Our verifier includes different test cases for domain, Table 11.1, and problem, Table 11.2. Some verifier tests refer to uncommon but valid PDDL, and can be seen as warnings for new users, such as actions with empty preconditions. Our verifier offers substantial help for novice users to understand their description mistakes by providing an automated analysis of the PDDL encoding.

Our validator applies each plan action, testing if such action exists (i.e., the action was defined and all parameters are defined objects/constants), is applicable (all positive preconditions are present in the current state, while no negative preconditions are present), and with their effects generate each intermediate state (current state with delete effects removed and add effects added). Note that the validator ensures simply that the provided plan is a solution to the problem, regardless of optimality, therefore empty and sub-step-optimal plans can also be used, as some problems may require no action, when an initial state satisfies a goal state, while other plans may even revisit intermediate states or simply take more steps than required using other action sequences. In this way, validators help domain engineers verify that their PDDL encoding allows a planner to generate valid plans, and that these plans indeed correspond to the intended semantics of the planning domain. Nevertheless, verifiers and validators tools are often separated from the actual planner software [12], which

[2] https://github.com/thlorenz/brace.

Table 11.1 Rules used by verifier in the domain description

Domain rule	Description
Predicate defined	Every predicate must be defined in *:predicates*
Predicate with valid name	Predicates must contain only valid characters, starting with *a-z*
Predicate arity	Predicates must have the same amount of parameters
Action redefined	Each action must have a unique name
Action parameter unused	One or more parameters of an action are unused
Action parameter repeated	One or more parameters of an action are repeated
Parameter with valid name	Parameters must contain valid characters, starting with *?*
Predicate repetition	Each predicate must appear only once in preconditions and effects
Empty precondition	Preconditions contain no predicates
Null effect	Effect is either empty or does modify state based on preconditions
Unnecessary equality	Preconditions contain *(= ?x ?y)*
Equality contradiction	Preconditions contain *(not (= ?x ?x))*
Precondition contradiction	Preconditions contain *(pre ?a)* and *(not (pre ?a))*
Effect contradiction	Effects contain *(pre ?a)* and *(not (pre ?a))*
Effect contains equality	Equality is only supported in preconditions
Missing/extra requirements	Requirements must match what is used in the description

Table 11.2 Rules used by verifier in the problem description

Problem rule	Description
Predicate repetition	Each predicate must appear only once in initial or goal states
Object with valid name	Objects must contain only valid characters, starting with *a-z*
Object unused	Objects must appear as constant terms in actions, initial, or goal states
Forced equality	Initial or goal states contain *(= a b)*
Goal contradiction	Goal state contain *(pre ?a)* and *(not (pre ?a))*
Rigid goal	Rigid goal predicate is unachievable unless present in initial state
Empty goal state	No planning is required for an empty goal state

makes reviewing and revising domain formalizations less straightforward. Thus, our coupling of the validator and verifier with the editor streamlines the domain formalization process by providing immediate feedback to the domain engineer.

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

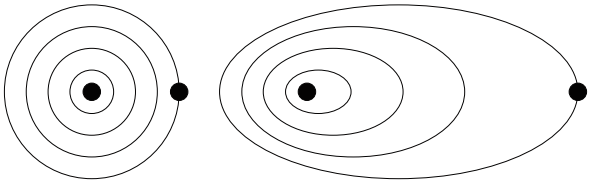

Fig. 11.3 Search contours are defined by search mechanism and heuristic function, either equally exploring in all directions (left) or giving priority towards the goal state (right)

Fig. 11.3, 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) [20, Chapter 3—page 97]. Such images target an audience new to the concept of using a computed auxiliary function to speed-up search. Different from textbooks, implementations that target the same audience use dynamic grids to show both how the state-space is explored and how the heuristic is computed in an Euclidean space. Such examples show 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, as they better represent state relations while ignoring repeated states by not expanding a previously found state. If we also added connections to previously found states, a cyclic graph would be obtained and the contours would not be visible.

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 more compact visualization. In practice, we use the Reingold–Tilford algorithm [18][3] 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 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 [7] as a heuristic. While we selected these two methods as examples to show the impact of no heuristic vs a generic distance metric for states, our visualization tool supports other search mechanisms and heuristic functions as long as such mechanisms search through the state-space.

To represent the data obtained from the planning process, we use a tree containing the explored state-space and heuristic information about each state. In this tree, each node represents a state (i.e., a set of instantiated predicates), an edge represents a state-transition (i.e., the execution of an action), and the root node represents the

[3]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.

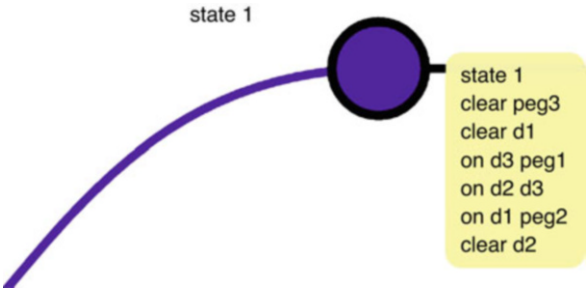

Fig. 11.4 Tooltip that displays the set of instantiated predicates in a state. This figure illustrates state 1 and its predicates for a planning instance of the Hanoi domain

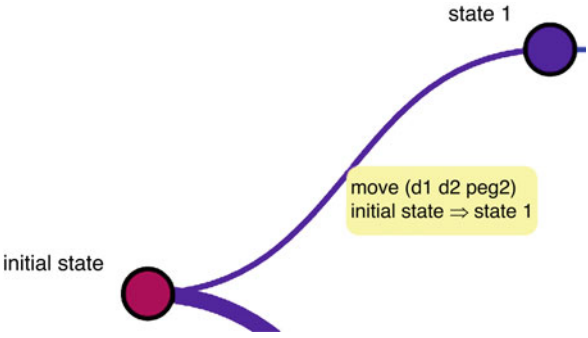

Fig. 11.5 Tooltip that displays the instantiated action applied between two states. This figure illustrates state 1 and its predicates for a planning instance of the Hanoi domain

initial state. The information about the set of predicates in the states (nodes) and the applied actions in such states (edges) are hidden in our heuristic visualization. Such information about states and actions can be seen when the user hovers the cursor over nodes and edges, which then shows, the state's and action's detail as a tooltip. Figures 11.4 and 11.5, respectively, illustrate how our visualization tool show the information about states and actions.

Our visualization tool displays the state-space of a planning heuristic by coloring the estimated distance between states using a heatmap, as in Fig. 11.6. Red nodes represent the states closer to the goal state, i.e., warmer, while distant nodes are represented by blue, i.e., colder. Nodes and edges are colored according to the estimated distance to the goal state. We illustrate the heuristic gradient as a heatmap in Fig. 11.7. Other heuristic functions could generate not only other distance estimations for each state (visible through colors in the graph), but also a different graph, as states would be explored in a different order, as in Figs. 11.10 and 11.11. Here, the radial layout of Fig. 11.11 provides a visualization of the search contours of the heuristic, provided a large enough sample of the total number of states has been explored. Edges between initial and goal state are emphasized (in bold, and as a thick line) to show which path contains the actions that constitute the plan. Such emphasized path is only available when planning is successful for the give planning task. Failed planning cases still obtain data to draw the explored state-space as a tree, which can be used as an interactive debug tool.

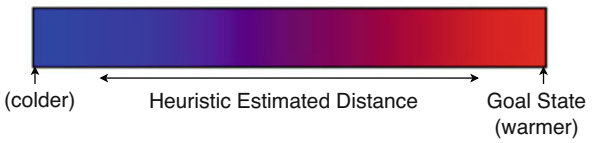

Fig. 11.6 Color scheme that our visualization tool uses to represent the estimated distance

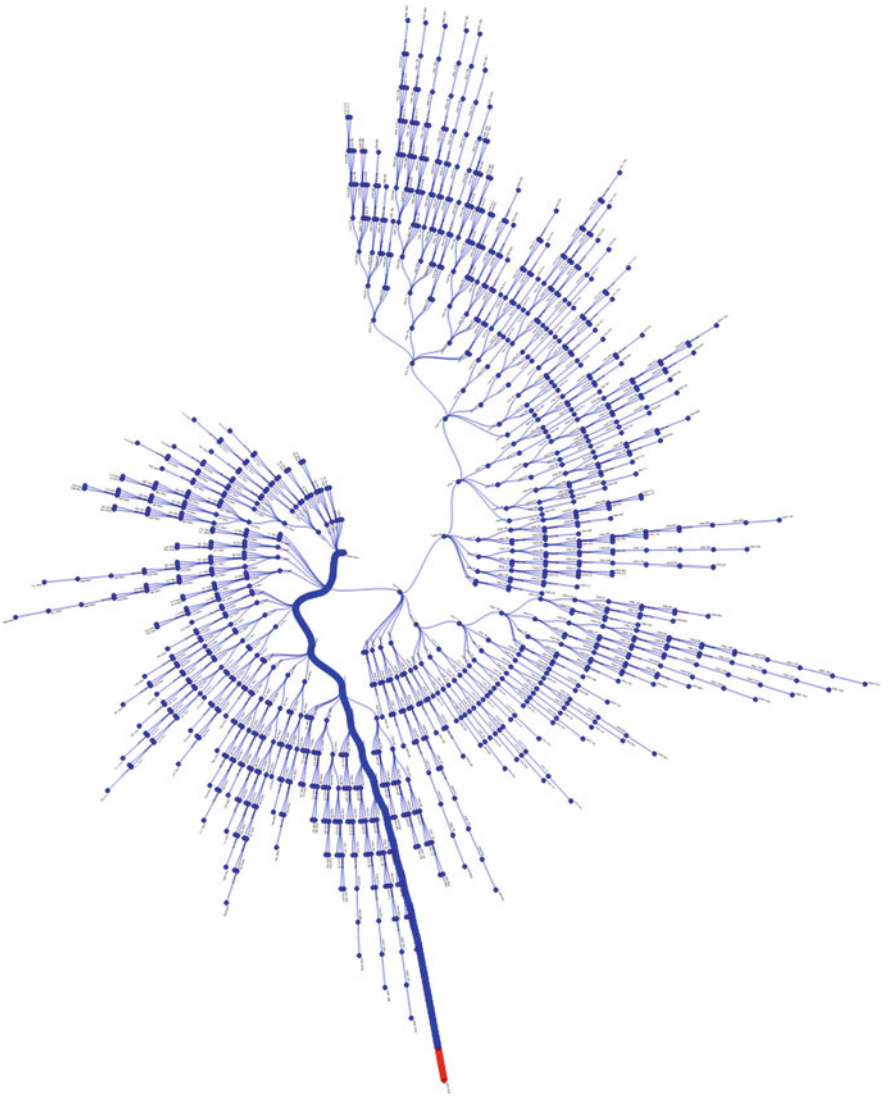

Fig. 11.7 Search contours become visible as more states are explored. This planning instance obtain all goal predicates at the same time, which makes the heatmap mostly blue (colder), while the goal state is located at the bottom in red (warmer)

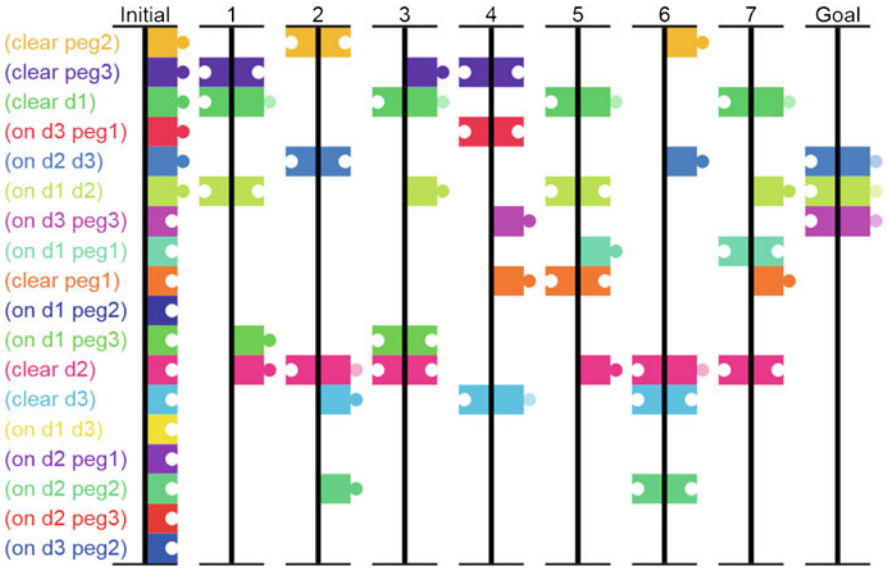

Fig. 11.8 Dovetail plan visualization of Hanoi domain with 3 discs and a plan of size 7

Fig. 11.9 Tooltip that displays the instantiated action in a plan on Dovetail

Dovetail Metaphor Visualization The second visualization we implemented is a visual metaphor called Dovetail [14], which is useful to see how predicates change along the plan execution. Each ground predicate that appears 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 Fig. 11.8), with tooltips providing extra information as shown in Fig. 11.9 for the domain of the case study of Sect. 4.1. The use of this visual abstraction (Dovetail) aims to improve the learning curve for defining and debugging planning domains and problems.

4 Deployment and Evaluation

4.1 Case Study

In order to validate our visualization tool, we now present a case study we carried out to show a planning instance using different planning heuristics displaying the state-space. To do so, we selected the Tower of Hanoi domain to illustrate our

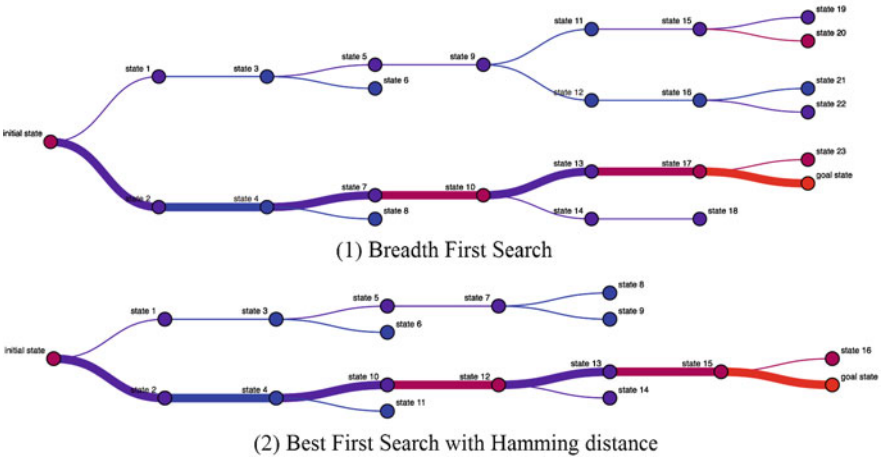

Fig. 11.10 Cartesian tree visualizations of the state-space of Hanoi with 3 discs

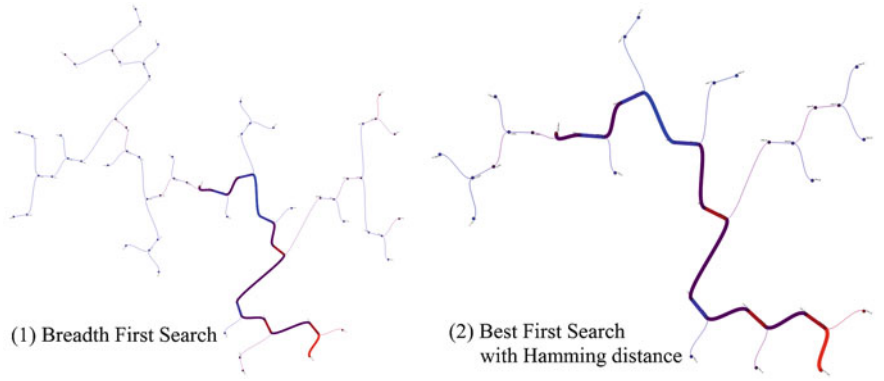

Fig. 11.11 Radial tree visualizations of the state-space of Hanoi with 3 discs

heuristic visualizations. In this domain, 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 for this domain show that the goal state cannot be achieved in an incremental way, requiring a plan to build and destroy partial towers several times, and then obtain the complete tower in the final peg. Domains with such particular 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 Fig. 11.10. The Cartesian tree generates a more compact representation, while the radial tree highlights the side to which the heuristic gave priority during search, as seen in Fig. 11.11, where the top-left branch was not explored. Other domains may suddenly achieve a goal state from a mostly blue colored graph, in which all states

are far away from the goal, as seen in Fig. 11.7, or incrementally achieving 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 [14] metaphor. This particular Hanoi planning instance is solved by a 7-step plan, represented by the pieces labeled with numbers at the top, Fig. 11.8. 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*.

4.2 Case Study Survey Results

To evaluate WEB PLANNER, a group of four users from our automated planning course[4] 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.[5] 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:
  - 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 Fig. 11.12 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

[4] https://github.com/pucrs-automated-planning/syllabus.

[5] https://ickeps2016.wordpress.com.

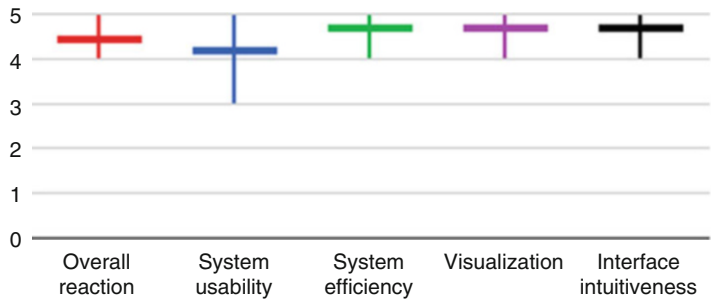

Fig. 11.12 Survey results, users were asked to evaluate the system between frustrating (0) and satisfying (5)

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.

4.3 General Public Usage Statistics

We collected anonymous data in WEB PLANNER from January 1 to May 30, 2019 to verify user habits. We identified users from multiple countries with varying session durations, with most users being in Brazil, where it was proposed as a classroom tool. World usage can be seen in Fig. 11.13

5 Related Work

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

`Planning.Domains`[6] is a collection of web tools for automated planning. These web tools provide a 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 debugging tools, such as TorchLight [10], and even WEB PLANNER visualizations as plugins. Similar to our approach, `Planning.Domains` provides a PDDL editor, however, our approach provides

[6] http://planning.domains.

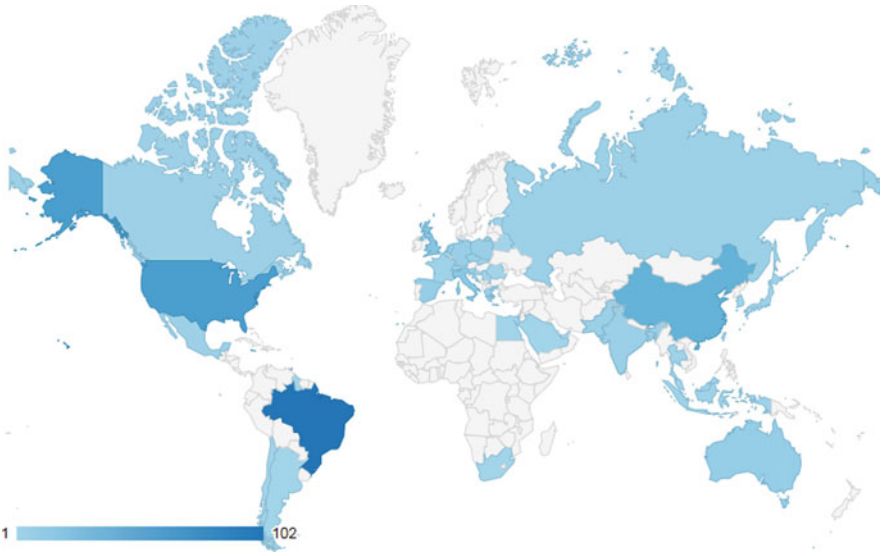

Fig. 11.13 User sessions per country during the first 4 months of 2019

not only a web editor with syntax highlighting, but also a set of tools to develop, analyze, and visualize planning domains using metaphors and alternative data visualization methods.

We consider two offline tools for PDDL file editing as related enough to our approach for comparison: `myPDDL`; and PDDL Studio. `myPDDL`[7] [22] is an editor extension for Sublime Text, which provides PDDL syntax highlighting, snippets, and domain visualization (e.g., diagram types). PDDL Studio [16] is an IDE to edit PDDL domains and problems. This IDE provides syntax highlighting, code completion, and context hints specifically designed for PDDL. Both tools have editor capabilities similar to ours, with `myPDDL` being able to generate type diagram and calculate distances automatically, two unique features that benefit only users that are either debugging typing errors or avoiding calculating distances in problems. PDDL Studio is able to list description errors and integrates with external planners using a command-line interface, leaving the user responsible for installation and call to each planner. While `myPDDL` and PDDL Studio are more flexible than our approach, being open or able to use any local planner, respectively, they need an initial setup phase that consumes valuable classroom time. One of our goals was to minimize the time spent to go from planning description explanation to planner call.

To validate a domain description one can follow the steps of a known valid plan to solve one problem and either achieve a goal state or discover errors in the domain

[7] https://github.com/Pold87/myPDDL.

description. An entire branch of plan validation tools was created from VAL[8] [12] to do this job automatically. More recent implementations, InVAL[9] and ReviVAL,[10] try to complement VAL, being independent implementations that can increase trust in domain descriptions and warn ambiguous PDDL descriptions to users. More PDDL validation tools means more interest in their usage in real-life activities, yet they are separated from planners and domain description tools. By adding a plan validator to our interface we expect to make not only the validation process simpler, but also essential to a user that wants an automated confirmation of their work, while bringing awareness that such tools exist.

Graphical Interface for Planning with Objects (GIPO) [21] is a tool for planning domain knowledge engineering that allows the textual specification of domains in PDDL and *Hierarchical Task Network* (HTN), like other code editors. 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. Like our approach, GIPO can use a set of plans to validate domain and problem specification, indicating whether the 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* [5] 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 [17] 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. *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. Our verification tests are only based on common PDDL mistakes and lack domain-dependent constraints.

itSIMPLE[11] [25] is concerned with domain modeling, using steps to guide the user from informal requirements (UML) to an objective representation (Petri Nets). *itSIMPLE* features 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. WEB PLANNER

[8] https://github.com/KCL-Planning/VAL.

[9] https://github.com/patrikhaslum/INVAL.

[10] https://github.com/guicho271828/ArriVAL.

[11] https://github.com/tvaquero/itsimple.

does not provide an incremental formalization approach to domain engineering, requiring users to start with PDDL descriptions and, once done, able to generate visualizations from it.

Magnaguagno et al. [14] developed a visual metaphor to help users visualize and learn how the planning process works. Dovetail results suggest that this visual metaphor can be useful to define and debug the planning process. We have applied this visual metaphor in WEB PLANNER by using colors each instantiated predicate in the state along a plan execution.

We found two approaches to data visualization suitable for heuristics. In [13], 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 [23] propose a set of strategies to visualize and compare changes in hierarchical data using treemaps. We currently only support state-space non-cyclic graphs obtained from the planning process and no graph comparison, as abrupt layout changes would impact a side-by-side comparison as perceived by Tu and Shen. We opted for the current tree structure to obtain a visible contour visualization that better matches abstract explanations.

6 Conclusions

In this paper, we describe WEB PLANNER, a cloud-based planning tool we developed that 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. Our small-scale survey indicated promising results with user-feedback pointing towards improvements and new features 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. WEB PLANNER has being used in the lectures from the Artificial Intelligence and Automated Planning courses since August 2016 to explain planning concepts using both PDDL and visualizations to around 50 students every year while being available to anyone online, reaching over 300 accesses in the first quarter of 2019.

We believe that such web tool can help new heuristics to be developed and tested, providing to users a better grasp of the impact of heuristics to the state-space exploration, which is usually an invisible entity. WEB PLANNER tool is available online at: https://web-planner.herokuapp.com.

Acknowledgements 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.

This research was achieved in cooperation with HP Brasil Indústria e Comércio de Equipamentos Eletrônicos LTDA using incentives of Brazilian Informatics Law (Law n° 8.2.48 of 1991).

Part of this research was also financed by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior—Brasil (CAPES)—Finance Code 001.

References

1. Bylander, T.: The Computational Complexity of Propositional STRIPS Planning. Journal of Artificial Intelligence Research (JAIR) **69**, 165–204 (1994)
2. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Journal of Artificial Intelligence Research (JAIR) **2**(3), 189–208 (1971)
3. Gerevini, A., Long, D.: 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 (2005)
4. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning—Theory and Practice. Elsevier (2004)
5. Glinskỳ, R., Barták, R.: VisPlan–Interactive Visualisation and Verification of Plans. Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) pp. 134–138 (2011)
6. Ha, T.T.: Theory and design of digital communication systems. Cambridge University Press (2010)
7. Hamming, R.W.: Error detecting and error correcting codes. Bell System Technical Journal **29**(2), 147–160 (1950)
8. Helmert, M.: The Fast Downward Planning System. Journal of Artificial Intelligence Research **26**, 191–246 (2006)
9. Hoffmann, J.: The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. Computing Research Repository (CoRR) **abs/1106.5271** (2011), http://arxiv.org/abs/1106.5271
10. Hoffmann, J.: The TorchLight Tool: Analyzing Search Topology Without Running Any Search. In: Proceedings of the System Demonstrations, in the 21th International Conference on Automated Planning and Scheduling. pp. 37–41 (2011)
11. Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research (JAIR) **14**(1), 253–302 (May 2001)
12. Howey, R., Long, D., Fox, M.: 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. pp. 294–301 (2004)
13. Kuwata, Y., Cohen, P.R.: Visualization Tools for Real-Time Search Algorithms. Computer Science Technical Report (1993)
14. Magnaguagno, M.C., Pereira, R.F., Meneguzzi, F.: DOVETAIL - An Abstraction for Classical Planning Using a Visual Metaphor. In: Proceedings of FLAIRS, 2016. (2016), http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS16/paper/view/12966
15. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL − The Planning Domain Definition Language. Technical Report – Yale Center for Computational Vision and Control (1998)
16. Plch, T., Chomut, M., Brom, C., Barták, R.: Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL Studio. In: Proceedings of ICAPS'09. pp. 15–18 (2012)

17. Raimondi, F., Pecheur, C., Brat, G.: 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 (2009)
18. Reingold, E.M., Tilford, J.S.: Tidier drawings of trees. IEEE Transactions on Software Engineering (2), 223–228 (1981)
19. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research (JAIR) **39**(1), 127–177 (2010)
20. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edn. (2009)
21. Simpson, R.M., Kitchin, D.E., McCluskey, T.L.: Planning domain definition using GIPO. Knowledge Eng. Review **22**(2), 117–134 (2007). https://doi.org/10.1017/S0269888907001063
22. Strobel, V., Kirsch, A.: Planning in the Wild: Modeling Tools for PDDL. In: Joint German/Austrian Conference on Artificial Intelligence. pp. 273–284. Springer (2014)
23. Tu, Y., Shen, H.W.: Visualizing Changes of Hierarchical Data using Treemaps. IEEE Transactions on Visualization and Computer Graphics **13**(6), 1286–1293 (Nov 2007). https://doi.org/10.1109/TVCG.2007.70529
24. Vallati, M., Chrpa, L., McCluskey, T.L.: What you always wanted to know about the deterministic part of the International Planning Competition (IPC) 2014 (but were too afraid to ask). Knowledge Engineering Review **33**, 383 (2018)
25. Vaquero, T., Tonaco, R., Costa, G., Tonidandel, F., Silva, J.R., Beck, J.C.: itSIMPLE 4.0: Enhancing the modeling experience of planning problems. In: Proceedings of ICAPS'12. pp. 11–14 (2012)
26. Ward, M.O., Grinstein, G., Keim, D.: Interactive Data Visualization: Foundations, Techniques, and Applications, Second Edition - 360 Degree Business. A. K. Peters, Ltd., Natick, MA, USA, 2nd edn. (2015)
27. Wilson, J.M.: Gantt charts: A centenary appreciation. European Journal of Operational Research **149**(2), 430–437 (2003)