From a Classroom to an Industry
From PDDL "Hello World" to Debugging a Planning Problem

Jan Dolejsi and **Derek Long** and **Maria Fox**
{JDolejsi, DLong6, MFox2}@slb.com
Schlumberger, UK
Christian Muise
christian.muise@ibm.com
IBM Research AI, Cambridge, USA

Abstract

This demonstration will build on previously shown end-to-end PDDL2.2 *developer environment* built on top of Microsoft's VS Code, but will expand substantially in two directions. First, the session from *editor.planning.domains* can seamlessly migrate from the web browser to VS Code editor and take advantage of more elaborate PDDL support. Second, VS Code now integrates debugging tools for PDDL modelers such as search tree visualization, step-by-step search debugger and plan validation. This offers coherent set of tools to use from classroom to an industrial planning application debugging.

1 Introduction

Since ICAPS 2018, the community started centralizing educational material on the *education.planning.domains* site, notably the Planning Domain Definition Language (PDDL Reference). However, for the success, it is important to consolidate and integrate such content and tools. For example the *editor.planning.domains* web-browser based experience has zero installation footprint and is therefore ideal as a classroom tool, but cannot support more advanced scenarios of interaction between the planning engine and the modeling environment. Such environment supporting rich interactions and deep integration into the planning engine is the (PDDL Extension for VS Code). This integrated developer environment for (PDDL2.2) (Edelkamp *et al.*, 2004) helps industrializing planning models (e.g. by declarative template-based problem file generation or visualizing the temporal and numeric aspects of plans), but also helps the modeler understand how is the planner looking for the solution and whether the PDDL model needs adjustments for better performance.

Why building such environment on top of VS Code? It was already demonstrated by (Brom *et al.*, 2012) or (Strobel Kirsch, 2015) that implementing a PDDL integrated developer environment from the ground up is feasible but hard to maintain over time. Furthermore, PDDL code remains isolated from the other components of the overall software solution.

With the drive to increased adoption of AI Planning and PDDL as the modeling language, this system demo proposes value-adding solution that seamlessly connects the classroom tools and more advanced industrial development tools into one experience, accelerates solution prototyping and makes planning problems diagnosable.

2 Guiding the PDDL novice

Starting in *editor.planning.domains* website, a novice learns by starting form a domain/problem example, invoking the cloud planner at *solver.planning.domains* and visualizing the plan. Number of plug-ins are available to streamline the experience by, for example, generating mundane parts of problem files.

2.1 From Web Browser to the Desktop

When they start building their own PDDL model and would benefit from a more elaborate PDDL support, they may save their progress into a session and with one click get the session open in VS Code, providing the environment is installed. The editor and the PDDL extension are available free of charge and runs on Windows, Linux and Mac.

2.2 Understanding PDDL Syntax

Once in the off-line editor, standard IDE features are available such as code snippets, auto-completion, hover-over tooltips, jump to declaration and find all references of a predicate, function or type across domain/problem/plan files. Syntax and validation errors and warnings are shown.

2.3 Introspecting into PDDL Plans

For ease of understanding of plans, plans are visualized using Gantt chart, resource usage swim lanes and line plots. (VAL) utilities are used to validate plans, flag errors and evaluate values of numeric functions.

2 Empowering the Expert

Beyond early prototyping, PDDL must be subject to the same continuous integration principles as any other code in the solution. The developer environment must therefore support at least version control and unit/regression testing.

2.1 Regression testing planning domains

To comprehensively test a planning domain, a growing number of problem files (and expected plans) must be maintained. As the solution grows and matures, this eventually becomes the bottleneck. It was demonstrated by (Building Support for PDDL as a Modelling Tool) that

adopting the declarative template-based modeling approach to the problem file assures integrity of every problem file (e.g. initialization of functions to zero) ever sent to the planner and ability to bulk-generate problem files from data. While the data for unit/regression test problem files may come from JSON files stored in version control, at runtime the data comes from sensors or databases. The advantage of this approach is that the exact same problem file template may be used in both scenarios.

Templating languages such as [Jinja2] or [Nunjucks] may be easily adopted by those familiar with JavaScript or Python respectively.

2.2 Understanding Planner Performance

While the goal of AI Planning is to furnish all industries with domain agnostic engine implementation that performs consistently on a wide range of PDDL domain encodings, the reality thus far is that some modeling approaches yield better performance than others given the engine implementation. Our experience has shown that PDDL modelers tend to omit obvious (to human) pre-conditions, build models with more-than-enough fidelity and do not appreciate how the size of the search space changes with modeling choices (including symmetry).

2.2.1 Search Graph Shape Indicator

Every note-worthy planning problem is searching for a solution in a space with too many dimensions to visualize for the human eye. However, there are certain characteristics of the search performance that could be visualized.

First indicator of the domain encoding efficiency is the general shape of the search tree. Search tree dominated by one long search branch is very efficient. Broad tree with too many short branches indicates missing pre-conditions. A combination (one long branch with one or more broad plateaus) indicates certain actions may need to be re-modeled to avoid the plateaus.

2.2.2 *Seen States* Summary Plots

While the planner is searching, it is discovering more states. Those *seen* states do not characterize the whole space, but indicate, whether and how fast is the planner closing the gap between the best state so far and the goal. States may be plotted in two ways: heuristic value vs state count and as a histogram of heuristic values.

2.2.3 Incremental State Visualization

Another visual that may be presented while the planner is searching is the best-state-so-far. The state could be depicted in 3 parts: the planhead (visualized as a Gantt chart), order of helpful actions and relaxed plan (as an approximate Gantt chart). As the search progresses, the planhead portion grows, while the relaxed plan shortens. When the planner reaches a plateau, the user can easily see what state is at the root of the plateau, which is an indicator for model improvement.

3. Conclusion

This system demo will present a range of PDDL learning and support tools from a web browser session to a powerful search debugger. Such toolset will improve the impact of classroom training as the participants will take home a working setup on their computer, which will empower them to build full blown planning solutions.

Figure 1 Rich plan visualization

```
{
    "defaultDomain": "trucks.pddl",
    "cases": [
        {
            "label": "Problem #1",
            "problem": "trucksp0.pddl",
            "expectedPlans": [
                "trucksp0-1.plan"
            ]
        },
        …
    ]
}
```

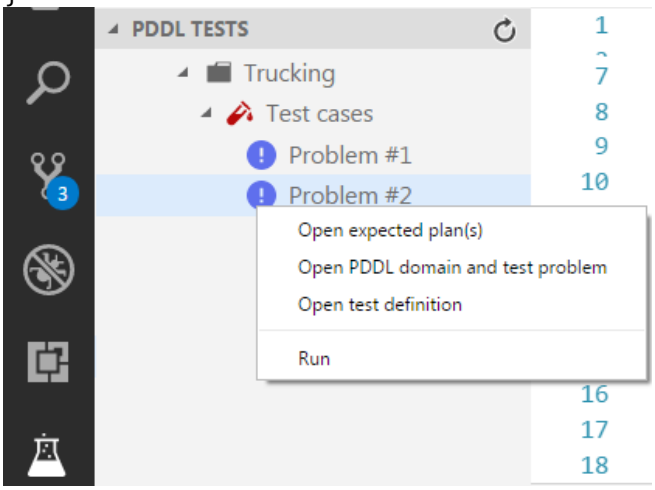

Figure 2 Unit/regression testing

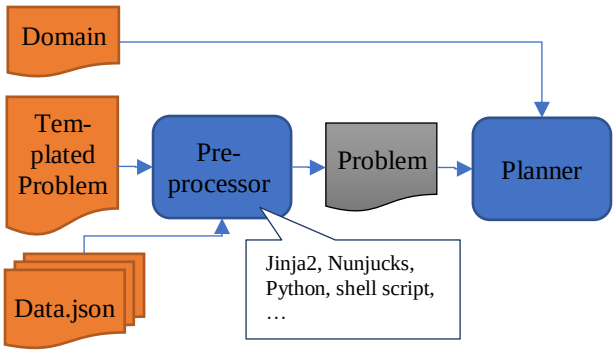

Figure 3 Templated problem file generation

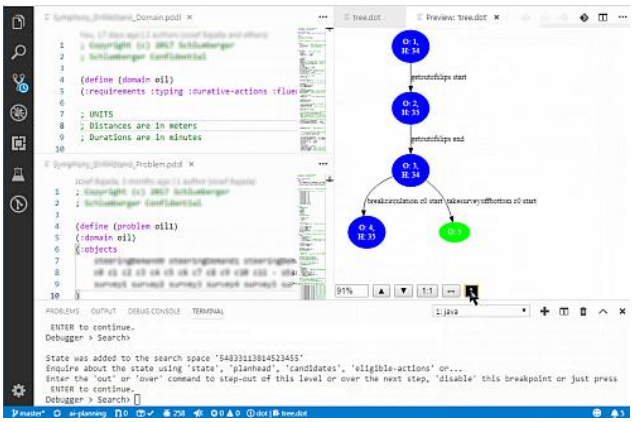

Figure 4 Visualizing search trees interactively

Figure 5 PDDL model efficiency characteristics and search debugging

References

[PDDL Reference] Adam Green: https://nergmada.github.io/pddl-reference/

[Building Support for PDDL as a Modelling Tool], ICAPS 2018, KEPS workshop, Derek Long, Jan Dolejsi and Maria Fox

[Edelkamp *et al.*, 2004] Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: the language for the classical part of the 4th international planning competition. Technical Report 195, ALU Freiburg.

[Brom *et al.*, 2012] Mgr. Cyril Brom Ph.D, PDDL Studio, https://amis.mff.cuni.cz/PDDLStudio/.

[Strobel Kirsch, 2015] Volker Strobel, Alexandra Kirsch, MYPDDL, https://www.researchgate.net/publication/284788212_Planning_in_the_Wild_Modeling_Tools_for_PDDL

[planning.domains, 2015] Andrew Coles, Christian Muise, Kristie Taylor-Muise, http://planning.domains/.

[VS Code] or [Visual Studio Code] Microsoft, https://code.visualstudio.com/.

[PDDL extension for VS Code], https://marketplace.visualstudio.com/items/jan-dolejsi.pddl/

[Jinja2] http://jinja.pocoo.org/docs/2.10/templates/

[Nunjucks] https://mozilla.github.io/nunjucks/

[parser] Parser configuration guidelines for PDDL VS Code extension: https://github.com/jan-dolejsi/vscode-pddl/wiki/Configuring-the-PDDL-parser

[planner] Planner configuration guidelines for PDDL VS Code extension: https://github.com/jan-dolejsi/vscode-pddl/wiki/Configuring-the-PDDL-planner