CS170 Fall 2019 Final Project:
Intractable Problems PDDL Solver
Team Name: Turing's Little Helpers

Carl Hentges
Vishnu Pamula
Emerson Shoichet-Bartus

December of 2019

1 Introduction

The goal of this project was to create a solver for the "Drive the TAs Home" problem (DTH problem). The approach taken by this team was to create a reduction from a DTH problem instance to a planning problem. More specifically, a problem formulated in the Planning Domain Definition Language (PDDL), which can then be interpreted and solved by a general purpose PDDL planner. PDDL planners have proven themselves capable of solving a variety of difficult problems, both in practice as well as in competitive problem solvers[1].

2 Approach

Solving an intractable problem by reducing it to another, better-understood problem is a common and well-understood technique in computer science. Intractable problems are often reduced to Boolean Satisfaction or Linear Programming problems; however, in this project we chose to reduce the DTH problem to a PDDL planning problem.

2.1 Technology: PDDL

Planning Domain Definition Language is a standardized Artificial Intelligence planning language, used to define a planning problem[2].A planning problem in PDDL consists of two parts. Firstly, a domain file which defines the environment in which the planner operates — more specifically, it defines actions that change the environment, based upon preconditions[1] and effects. Additionally, the domain can also contain functions, such as those used to describe the cost of a solution. The second part of a planning problem is the problem instance, this defines the variables, initial conditions and goal state. The planner (solver) takes these two files and creates a plan, describing the sequence of actions to take in order to get from the initial state to the goal state.

[1]Conditions that must be true before an action can be applied; for instance, a TA must be at location A and there must be a path from A to B for the TA to walk from A to B, and A and B cannot be the same location

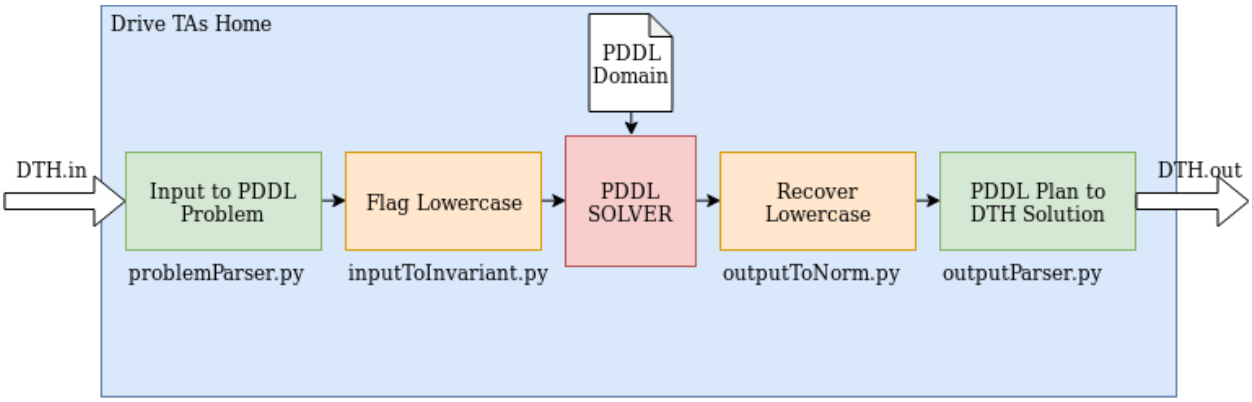

Figure 1: Diagram of the Reduction from DTH problem to PDDL planning problem.

2.2 Methodology

2.2.1 Identification of the required tasks

When first approaching the problem, several steps were identified that would need to be solved in order to enable the solving of the DTH problem.

1. Translation from problem instances in DTH to PDDL problem instances.
2. Translation from PDDL solutions to DTH solutions.
3. A PDDL domain describing the actions that could be applied in a DTH problem.
4. An automated way of applying these steps to the large number of inputs.
5. The identification of a suitable planner to solve this problem.

2.2.2 Step 1 & 2: Translation to and from PDDL

A series of programs were written in Python[2] in order to translate from DTH to PDDL and back again. The programs responsible for parsing the DTH problems and then parsing the output of the PDDL solver are `problemParser.py` and `outputParser.py`, respectively. Some issues were identified at this stage, namely PDDL is case-invariant, i.e. it cannot differentiate between upper and lowercase characters, but the output of DTH is case-sensitive. To solve this, another pair of programs were created. The first, `inputToInvariant.py` takes the output of `problemParser.py`, and adds a "flag" to every lowercase character. The second, `outputToNorm.py` then takes the PDDL plan and removes the "flags", and recovers the lowercase characters, before passing the file on to `outputParser.py`. A diagram is provided to help clarify the different stages of this process (Fig.1).

2.2.3 Step 3: Creating the PDDL Domain

The PDDL domain was created manually based on the problem specification. Once the domain was created, it was not necessary to change it since it remained the same across all permutations of the DTH problem instances.

[2]Python was chosen as it is an easy-to-use language, which facilitated the rapid creation of the software for this project. Additionally, the primary time bottleneck for solving the inputs was the PDDL solver, so it would not have made a significant time difference to use a low level language (e.g. Java or C)

2.2.4 Choice of PDDL Planner

Two PDDL planners were used in this project.The first, `at_BFS` (Anytime Best First Search), is an anytime planner, meaning it will continue to seek better solutions to a problem, until it can no longer improve the solution quality. This works well for small inputs, but fails on larger problem instances. The second planner used, was `BWFS` ([Dual] Best Width First Search), a modern competition planner, which was runner up in the latest International Planning Competition — IPC 2018[3][4]. This planner worked better on the larger problem inputs. However, it does not optimize the solutions found, and thus generally had a worse solution quality when compared to the solutions found with the `at_BFS` planner. Both of these planners ran on the Lightweight Autonomous Planning toolKiT (LAPKT), a framework that supports the running and creation of PDDL planners[5].

3 Results

Initial results for this approach to solving the problem were promising, solving small problem instances such as the example problem given in the problem description optimally. While this approach worked reasonably well on the simple, size-50 problem instance, as the sizes and complexities of the problem increased, the planner used considerably more memory[3] and time. As a result of this, the solution quality and quantity suffered in the larger problem instances.

4 Conclusion and Discussion

In this report, we demonstrated a reduction from DTH problem to a PDDL planning problem, and showed that for small examples it generated reasonably good results, but at larger input sizes the increase in computation time limited the production of high quality solutions.

5 Further Research

The major limitation of this approach is the state-space-explosion; the need for a large amount of processing power, and especially memory, needed to cope with this. Barring changes to the planner, one potential way to mitigate this is to use a cloud computing platform, such as Amazon Web Services (AWS), which would allow the solver to access a far greater amount of memory and processing power than a local machine.

References

[1] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[2] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[3] Nir Lipovetzky and Hector Geffner. A polynomial planning algorithm that beats lama and ff. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[4] Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[5] Miquel Ramirez, Nir Lipovetzky, and Christian Muise. Lightweight Automated Planning ToolKiT. http://lapkt.org/, 2015. Accessed: 2019-12-6.

[3] Regularly using 16+ GB of memory.