Universitat Politècnica de Catalunya
Universitat de Barcelona
Universitat Rovira i Virgili

Master in Artificial Intelligence

Planning and Approximate Reasoning

The Coffee Problem

Authors:
Emanuel Sanchez Aimar
Johannes Heidecke

November 2016

Contents

List of Figures

1 Introduction

This deliverable uses STRIPS (Stanford Research Institute Problem Solver) to solve the following linear planning problem:

> *There is a squared building composed by 36 offices, which are located in a matrix of 6 rows and 6 columns. From each office it is possible to move (horizontally or vertically) to the adjacent offices. The building has some coffee machines in some offices that can make 1, 2 or 3 cups of coffee at one time.*
>
> *The people working at the offices may ask for coffee and a robot called "Clooney" is in charge of serving the coffees required. Each office may ask for 1, 2 or 3 coffees but not more. The petitions of coffee are done all at early morning (just when work starts) so that the robot can plan the service procedure. Each petition has to be served in a single service.*
>
> *The goal is to serve all the drinks to all the offices in an efficient way (minimizing the travel inside the building, in order to not disturb the people working).*

2 Problem Analysis

To analyze this problem thoroughly, we describe the used predicates (subsection 2.1), the possible initial and goal states (subsection 2.2), the operators (subsection 2.3), the search space (subsection 2.4) and special cases (subsection 2.5).

2.1 Predicates

The used predicates are the following:

- `Robot-location(o)`: The robot is in office o. We chose a coordinate based representation of offices where each o is defined by its coordinates $[x, y]$.
- `Robot-free`: The robot does not have any coffee loaded.
- `Robot-loaded(n)`: The robot has n cups of coffee loaded with $n \in \{1, 2, 3\}$.
- `Petition(o, n)`: n cups of coffee are required in office o with $n \in \{1, 2, 3\}$.
- `Served(o)`: Office o has been served.
- `Machine(o, n)`: There is a coffee machine in office o that produces exactly n cups of coffee with $n \in \{1, 2, 3\}$.
- `Steps(x)`: The robot has moved x steps with $x \in \mathbb{N}_0$.

2.2 Intitial and Goal States

The **initial state** of any problem is composed as follows:

- **Required predicates**: *Robot-free*, *Steps(0)*, *Robot-location(o)* where o is a valid office location.
- **Optional predicates**: One or more *Petition(o,n)* with at most one petition for each office.
- **Conditionally required predicates**: For each $n^* \in \{1, 2, 3\}$ that occurs in a $Petition(o, n^*)$, there has to be at least one *Machine(o, n)* with fitting $n = n^*$ in order to guarantee that the problem is solvable.

The **final state** of any problem is composed as follows:

- **Optional predicates**: A *Robot-location(o)* can be specified.
- **Conditionally required predicates**: For each o^* that occurs in a $Petition(o^*, n)$, there must be exactly one $Served(o^*)$ predicate in the final state.

2.3 Operators

This problem contains three operators that can be applied to reach the final state starting from the initial state. These operators are: $Move(o_1, o_2)$, $Make(o, n)$ and $Serve(o, n)$.

Move:

The operator $Move(o_1, o_2)$ allows the robot to move between offices and is set up in the following way:

- **Preconditions**: Robot-location(o_1), Steps(x).
- **Add**: Robot-location(o_2), Steps($x + distance(o_1, o2)$).
- **Delete**: Robot-location(o_1), Steps(x).

Make:

The operator $Make(o, n)$ allows the robot to make coffee in an office with a machine and is set up in the following way:

- **Preconditions**: Robot-location(o), Robot-free, Machine(o, n).
- **Add**: Robot-loaded(n).
- **Delete**: Robot-free.

Serve:

The operator $Serve(o, n)$ allows the robot to serve coffee and is set up in the following way:

- **Preconditions**: Robot-location(o), Robot-loaded(n), Petition(o, n).
- **Add**: Served(o), Robot-free.
- **Delete**: Robot-loaded(n), Petition(o, n).

2.4 Search Space

The search space of a problem is the subset of the state space that can be reached by applying the operators, starting at the initial state. The nodes in the search space correspond to states of the world, while the edges correspond to state transitions through operators.

To compute the size of the search space we have to consider how many states we can reach from a given initial state by only applying our operators. In order to compute the size of the search space we take the following considerations into account:

- The predicate Robot-location(o) is always present exactly one time in each state, since it is required in the initial state and no operator adds this predicate without deleting a different instantiation of it and vice versa. There are 36 different values for o (one for every office in the world.
- Robot-free and Robot-loaded(n) cannot be part of a state at the same time (see Make and Serve operators). Robot-loaded(n) can take three different values: $1, 2$ and 3. This leads to 4 different combinations: Robot-free, Robot-loaded(1), Robot-loaded(2) and Robot-loaded(3).
- The Petition(o, n) predicates are taken from the initial state and can then be removed via the Serve operator. Since each Petition predicate can either still be present or replaced with a Served predicate, the number of possible states is $2^{|P|}$ where $|P|$ is the number of petitions in the initial state.
- The predicate Steps(x) is only present once, but x can take a arbitrary integer value larger than 0.

The Steps(x) predicate makes the search space infinitely large. Since our goal is to both reach the final state while minimizing x, we will consider the search space ignoring x and then later twist our algorithm in a way to explore the search space with consideration to keeping x small.

Ignoring the Steps(x) predicate, our search space is of size $36 \cdot 4 \cdot 2^{|P|}$ with $|P|$ being the number of petitions in the initial state. The size of the search space grows exponentially with the number of petitions.

2.5 Special Cases

Some combinations of initial states and final states are not able to be solved. This is the case when the final state is not part of the search space. An example case for this scenario is a Petition(o, n_p) predicate that does not have a matching Machine(o, n_m) with $n_p = n_m$.

3 Planning Algorithm

3.1 STRIPS Algorithm

The implementation is based on the STRIPS algorithm which works in the following way:

3.2 Heuristics

4 Implementation

4.1 Class structure

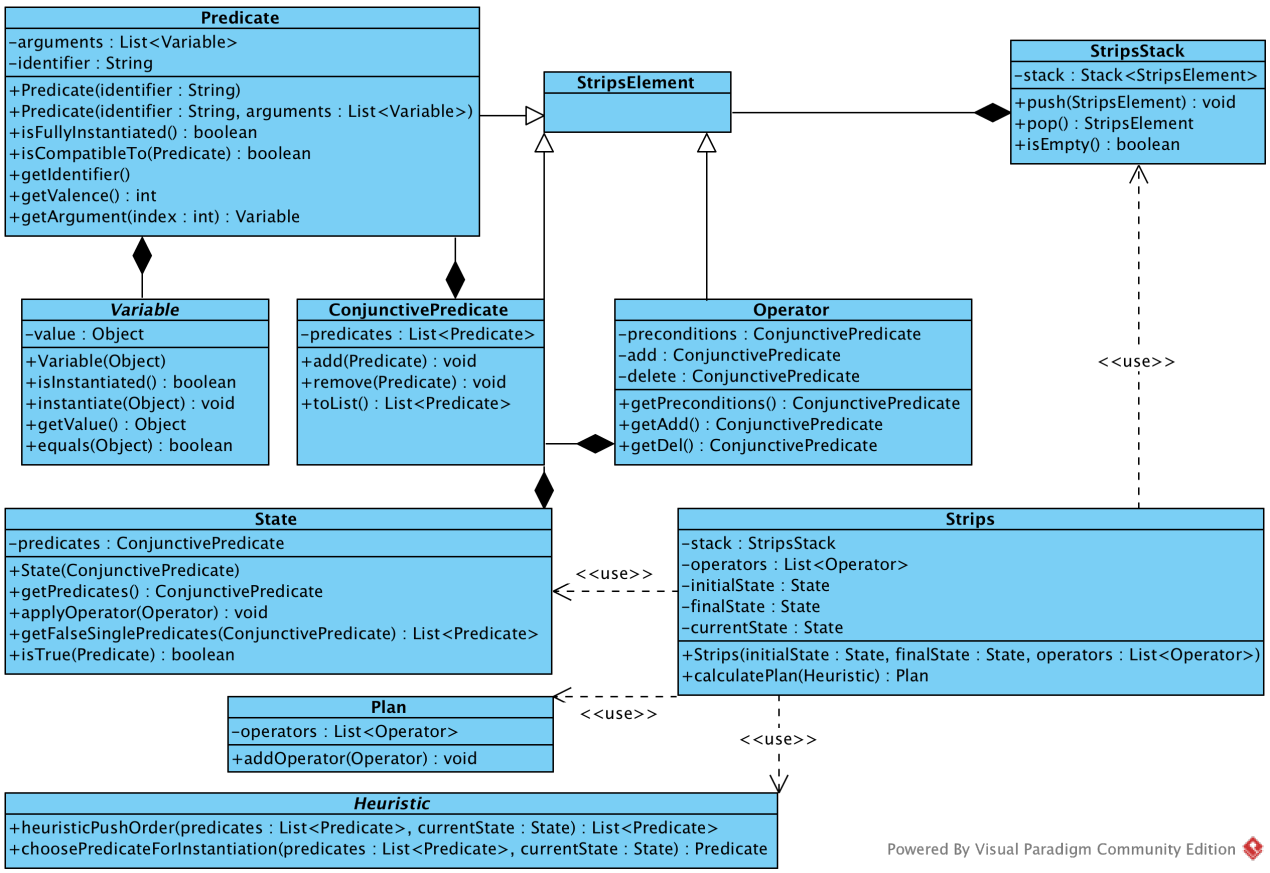

Figure 1: UML Class Diagram of STRIPS classes

4.2 Implementation details

Listing 1: Strips method *instantiate*

```
private void instantiate(Predicate singlePred, Heuristic heuristic) {
  List<Predicate> compatiblePredicates = new ArrayList<>();
  for (Predicate currPred : this.currentState.getPredicates().toList()) {
    // find all compatible predicates in current state:
    if (currPred.isCompatibleTo(singlePred)) {
      compatiblePredicates.add(currPred);
    }
  }
  ...
  // Heuristic chooses one candidate from compatiblePredicates
  Predicate chosenPred = heuristic.choosePredicateForInstantiation(
                                compatiblePredicates, currentState);

  // instantiate with singlePred with constants of chosenPred
  for (int i = 0; i < chosenPred.getValence(); i++) {
    if (!singlePred.getArgument(i).isInstantiated()) {
      // instantiate updates the java object of the variable.
      // all references to this variable in other predicates
      // will now reference to the instantiated variable.
      singlePred.getArgument(i).instantiate(
                    chosenPred.getArgument(i).getValue());
    }
  }
}
```

5 Tests

6 Results

results (show the contents of the stack during the execution, not only the final path). Analysis of the results (graphics with complexity, number of steps, etc.).

Appendices

A Included Files

B Eclipse Instructions

Instructions to execute the program.