TÉCNICO LISBOA

Artificial Intelligence and Decision Systems (IASD)
Mini-projects, 2016/2017

Assignment #2

Version 1.1 - November 25st, 2016

SATPLAN

1 Introduction

Scientific competitions are a powerful drive for the advancement of research in many areas, including AI, Robotics, and Computer Vision. In these competitions, different approaches to the same problem are evaluated and compared in a systematic way, awarding the best performing ones. In the area of AI, ICAPS International Planning Competition is being held since 1998, where planning algorithms compete for the fastest approach solving a set of benchmark planning problems[1].

Most of the state-of-the-art planners in these competitions use highly effective heuristic search. However, it came by a surprise that in 2004, SATPLAN, a planner based on a rather different approach outperformed all the other planners in optimal STRIPS planning. SATPLAN is based on the idea of first encoding the planning problem into a SAT problem, and then solving it using a state-of-the-art SAT solver. This approach was first proposed by Kautz in 1992 [4], but it took more than a decade for it to become competitive, after significant improvements in both SAT solving and SAT encoding of planning domains.

The goal of this mini-project is to develop a planner using the SATPLAN approach. Both a PDDL to SAT encoder and a SAT solver have to be developed. This assignment suggests the use a linear encoder together with a DPLL solver to solve PDDL problems. However, the students are strongly encouraged to adopt alternative/improved encoding and SAT solving methods.

2 Problem statement

PDDL is a planning domain description language building upon First Order Logic (FOL). In PDDL all terms are either variables or constants, that is,

[1] `http://icaps-conference.org/index.php/main/competitions`

no functions are allowed. We will use the blocks world as a running example throughout this assignment. A PDDL[2] domain is defined by three components:

- an initial state, defined as a conjunction of ground[3] atoms, e.g.,

$$on(A, B) \wedge on(B, C) \wedge on(C, Table) \wedge clear(A)$$

- a goal state, also defined as a conjunction of ground atoms, e.g.,

$$on(C, B) \wedge on(B, A) \wedge on(A, Table) \wedge clear(C)$$

- and a set of action schemas, each one comprising the action name and arguments (using the same syntax as atoms), the preconditions and the effects, each of these defined as a conjunction of literals, e.g.,

  Action($Move(b, f, t)$,
  PRECOND: $on(b, f) \wedge clear(b) \wedge clear(t)$,
  EFFECT: $\neg on(b, f) \wedge on(b, t) \wedge \neg clear(t) \wedge clear(f)$)

  Action($MoveToTable(b, f)$,
  PRECOND: $on(b, f) \wedge clear(b)$,
  EFFECT: $\neg on(b, f) \wedge on(b, Table) \wedge clear(f)$)

  For the simplicity sake, we will assume that all variables in the precondition and effects are included in the action arguments.

Notation: we will use capitalized words for constants and action names, and uncapitalized words for variables and atoms.

The goal of a planning algorithm is to determine the shortest sequence of ground actions — the *plan* — such that, after being applied in sequence from the initial state on, the goal state is satisfied. For the above case, the shortest plan is

$$[\, MoveToTable(A, B),\ Move(B, C, A),\ Move(C, Table, B)\,]$$

SAT stands for a boolean satisfiability. A SAT problem consists in determining a True/False assignment to the set of boolean variables of a propositional sentence in CNF[4]. An example of a SAT problem is

$$x_1 \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

A SAT problem can be either *satisfiable* or *unsatisfiable*, depending on whether or not a variable assignment exists satisfying the corresponding sentence. In

[2] We will adopt the same formalism as the course textbook [5].
[3] A ground atom is an atom whose terms are all constants.
[4] Conjunctive Normal Form, that is, a conjunction of disjunctions of literals.

this example, the variable assignment $\{x_1 = T, x_2 = F, x_3 = T\}$ makes this sentence satisfiable. There is a large variety of computer science problems that can be cast as a SAT problem. Thus, advances in the area of SAT solving have a broad impact.

SATPLAN is a planning method consisting in interleaving (1) the encoding of the planning problem as a SAT sentence, with (2) solving the result with a SAT solver. Encoding is performed for a given amount of time steps, called the *horizon*. For an horizon of h steps, if the resulting SAT sentence is satisfiable, then the planning problem has a solution with exactly h steps. To obtain the shortest plan, the encoding is iteratively performed for $h = 1, 2, \ldots$ until the corresponding SAT sentence is satisfied. The satisfiable SAT sentence with the shortest horizon is the shortest solution to the planning problem.

One of the simplest encoding of a PDDL problem to a SAT sentence is the *linear encoding,* described here. First, all actions should be grounded, that is, each action schema is replicated for all possible substitutions of variables by constants in the domain. In our example, the set of constants in the domain is $\{A, B, C, Table\}$ and the action $Move(b, f, t)$ is transformed into $4^3 = 64$ ground actions[5]. The set of all ground atoms in the initial and goal states, as well as in the preconditions and the effects of all actions, is called the *Hebrand base.*

Second, the SAT variables will include both the grounded atoms and the actions names and arguments, indexed by time. For instance, variable $MoveToTable(A, B)_1$ set to true means that the action $MoveToTable(A, B)$ was performed at time step $t = 1$, while variable $on(A, B)_0$ set to true means that $on(A, B)$ holds for time step $t = 0$.

For a given time horizon h, the resulting SAT sentence is the conjunction of the following parts:

1. the initial state holds for the initial time step $t = 0$, e.g.,

   $$on(A, B)_0 \wedge on(B, C)_0 \wedge on(C, Table)_0 \wedge clear(A)_0$$

   while all other atoms in the Hebrand base are explicitly negated[6], e.g.,

   $$\neg on(A, A)_0 \wedge \neg on(A, C)_0 \wedge \neg on(A, Table)_0 \wedge \cdots$$

2. the goal state holds for the final time step of the horizon, $t = h$, e.g.,

   $$on(C, B)_3 \wedge on(B, A)_3 \wedge on(A, Table)_3 \wedge clear(C)_3$$

   for $h = 3$;

[5] In fact, some of them are useless, such as $Move(A, A, A)$, but they will never be part of any output plan. This could be avoided by adding preconditions constraining the variables to be different, e.g., $b \neq f$, but for the simplicity sake this option will not be considered here.

[6] This is a consequence of the Closed World Assumption taken by PDDL.

3. actions imply both their preconditions and their effects, for all time steps $t = 0, \ldots h-1$, e.g., for $t = 0$,

$$MoveToTable(A,B)_0 \Rightarrow on(A,B)_0 \wedge clear(A)_0 \wedge \neg on(A,B)_1 \wedge on(A,Table)_1 \wedge \neg clear(B)_1$$

which can then be easily translated into CNF;

4. frame axioms, stating that, for each ground actions and for each time step $t = 0, \ldots h-1$, any atom in the Hebrand base not modified by an action maintains the same logical value from t to $t+1$, e.g., for $t = 0$,

$$on(B,C)_0 \wedge MoveToTable(A,B)_0 \Rightarrow on(B,C)_1$$
$$\neg on(B,C)_0 \wedge MoveToTable(A,B)_0 \Rightarrow \neg on(B,C)_1$$

5. and finally, exactly one action is performed in each time step $t = 0, \ldots h-1$, e.g., for $t = 0$,

$$Move(A,A,A)_0 \vee Move(A,A,B)_0 \vee Move(A,A,C)_0 \vee \ldots$$
$$\neg\left(Move(A,A,A)_0 \wedge Move(A,A,B)_0\right) \wedge$$
$$\neg\left(Move(A,A,A)_0 \wedge Move(A,A,C)_0\right) \wedge$$
$$\ldots$$

also known as the at-least-one and the at-most-one axioms.

All of these parts are then put together into a single SAT sentence to be fed to the SAT solver.

This is one of the simplest methods to encode planning problems in SAT. Several improvements have been proposed in the literature leading to more efficient encodings (see for instance [2]). Groups are strongly encouraged to implement one or more of these improvements.

For the SAT solver, we propose the use of the well-known DPLL algorithm, with the improvements each group is also strongly encouraged to implement (see for instance [3]). From the solution, the resulting plan can then be easily extracted by checking the logical value of all variables corresponding to action names. In the case of the running example, the result would be a satisfiable sentence with minimum $h = 3$, having the action variables $MoveToTable(A,B)_0$, $Move(B,C,A)_1$, and $Move(C,Table,B)_2$ set to true, while all other action variables set to false.

3 Implementation requirements

The planning domain is specified in an input text data file. For the simplicity sake we will adopt a significantly simpler syntax than PDDL, while retaining the relevant features for this mini-project.

Atoms have the format

name`(`*term1*`,...)`

where a term is either a constant or a variable, represented with capitalized and uncapitalized words, respectively. No spaces are allowed inside an atom. A literal is an atom, that, if negative, is prefixed with a '`-`' character.

Each line of the input data file may either be blank or specify one of the following items:

- the initial state, with the format:

  `I` *atom1 atom2 ...*

  defining a conjunction of the *atom*'s (all ground). The file must contain at most one such line.

- the goal state, with the format:

  `G` *atom1 atom2 ...*

  defining a conjunction of the *atom*'s (all ground). The file must contain exactly one such line.

- the action schemas, one per line, with the format:

  `A` *name* `:` *precond1 ...* `->` *effect1 ...*

  where *name* is an atom and *precond*'s and *effect*'s are literals. This defines an action with name *name*, preconditions defined as the conjunction of *precond*'s and the effects of *effect*'s.

All other lines should be ignored. Lines may appear in any order.

As an example, the blocks world can be represented in the following way:

```
I on(A,B) on(B,C) on(C,Table) clear(A)

A move(b,f,t) : on(b,f) clear(b) clear(t) -> -on(b,f) on(b,t) -clear(t) clear(f)
A move2table(b,f) : on(b,f) clear(b) -> -on(b,f) on(b,Table) clear(f)

G on(C,B) on(B,A) on(A,Table) clear(C)
```

The project is to be implemented in the Python programming language (version 3) as a program accepting as argument the filename of the problem specification, as described above. For instance, to solve the blocks problem one may run[7]

```
python satplan.py blocks.dat
```

assuming that the above file is stored in the current directory with filename `blocks.dat`. The name of the program `satplan.py` is arbitrary.

The output of the program to the display should be a sequence of ground action names and arguments, one per line, following the same syntax of the input file. One output example of the solution to the blocks problem is:

[7] Assuming that the default python command is version 3.

```
move2table A B
move B C A
move C Table B
```

4 Assignment goals

1. Develop a planner in Python (version 3) following the SATPLAN method described above. The planner should be called from a program taking the domain data file as single argument.

   In the code, the SAT solver part *should be* explicitly isolated from the rest. In particular this should be done using different Python files, one or more for the SAT solver and one or more for the rest. Use self-evident filenames for the Python files.

   Suggestion: Use the DIMACS syntax, described in section 2.1 of [1], when feeding SAT problems to the solver. This will allow for both efficient SAT solving and straightforward testing/comparison with state-of-the-art SAT solvers (such as zChaff and Minisat).

   Note: All code should be adequately commented.

2. The answers to the following points should be included in a short technical report, delivered together with the source code.

   (a) Describe the SAT encoding used, if different than the one described here.

   (b) Describe all improvements done (if any) over the basic DPLL algorithm.

   (c) Evaluate quantitatively the benefits of the improvements introduced to the SAT solver (if any).

3. The deliverable of this Lab Assignment consists of a ZIP file containing:

   - all the Python source code, and
   - the short report (in **PDF** format) with no more than 2 pages.

Submission: electronic submission via *fenix*[8].
Deadline: 23h59 of 10-Dec-2016
(Projects submitted after the deadline will not be considered for evaluation.)

[8] Go to *Student* > *Submit* > *Projects* after login.

References

[1] Satisfiability suggested format. `http://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/satformat.ps`, 1993.

[2] Michael D Ernst, Todd D Millstein, and Daniel S Weld. Automatic SAT-compilation of planning problems. In *Proceedings of teh 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1169–1176, Nagoya, Japan, 1997.

[3] Carla P Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, chapter 2, pages 89–134. Elsevier, 2008.

[4] H. A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of ECAI-92*, pages 359–363, 1992.

[5] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.