Graph-based Planning

STRIPS planning based on graphs

Michele Lombardi <michele.lombardi2@unibo.it>

Exercise

Start Graphplan

- Graphplan can run from the command line:

  gigi@lab2:~$ `graphplan -h`

- From the course website, download the "cart_example_gp.zip" archive:

  ```
  operator files: cart.gp.ops
  Fact file: cart.gp.facts
  ```

- In case you want to do some exercise at home, you can find the executable graphplan (for Linux!) on the course website.

A classic example

1. Two subjects

2. A starting place

3. A means of transport

4. A destination place

Exercise

Construction of the graph…

- Try to solve the truck problem with Graphplan
- Gradually increase the level of output detail, using the "-i" option
- Try to identify:
  - The graph gradually produced
  - The lists of mutual exclusion
  - The time point when the search for a valid plan starts
- Try to disable the use of the mutual exclusions

FF: Fast Forward

Planning based on graphs as a heuristic

Exercises

Start FF

- On the laboratory PCs also Fast Forward is installed
- Try to run it on the example of the trolley with:

```
gigi@lab2:~$  ff -f <dataFile>
                  -o <operatorFile>
```

- Try running the program using as input the two .pddl and and .facts files from the truck example

SATPLAN & Blackbox

Unify planning based on graphs and planning as satisfiability

SATPLAN & Blackbox

An observation

- Graphplan is looking for a valid plan by branching on sets of propositions
- If the problem is simple, this saves several branches and a plan is found quickly
- If the problem is complex, however, the number of sets on which to branch can become very big...

One idea (Kautz & Selman, '99)

- Model the search for a valid plan as a SAT problem (SAT stands for satisfiability)
- BLACKBOX planner

SATPLAN & Blackbox

SATsfiability problem:
Decide if a logical formula is satisfiable

Declarative approach:

1. We model a problem by:
   - logical variables (0-1 domain)
   - operators and, or, not
2. Using a solver generic to find an assignment of the variables that satisfies all the constraints

Why SAT?

Because there are very efficient solvers for SAT that would become usable as planners

SATPLAN & Blackbox

In SAT:

- logic variables (0-1)
- constraints = disjunctions of variables (positive or negated)

Example:

"If the weather is bad then I go out by car"

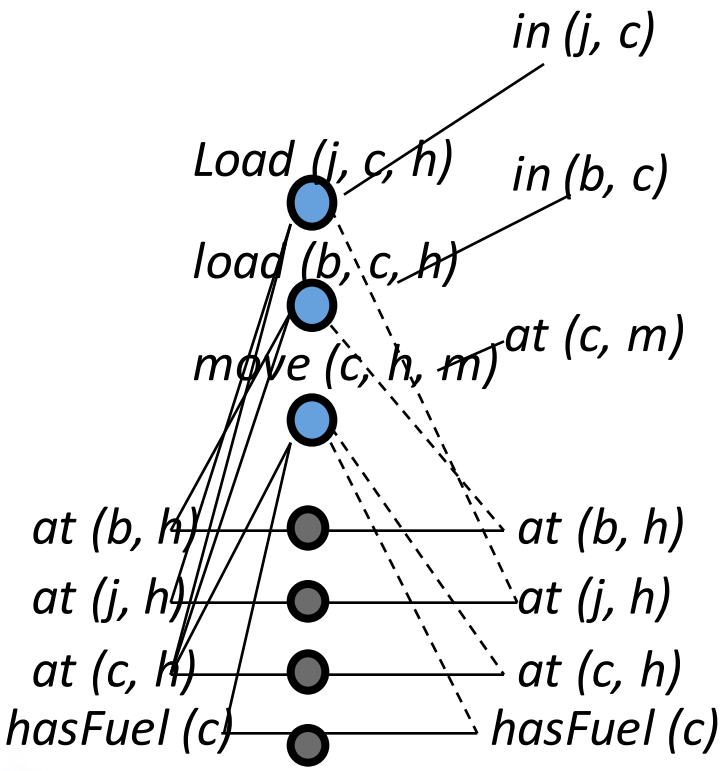
X_0 X_1

- Only one constraint: $X_0 \rightarrow X_1$
- Using only disjunctions and negations: **not** X_0 **or** X_1
- The use of the conjunction is implied (specify more constraints)
- All constraints are a single logical formula (Well Formed Formula)

SATPLAN & Blackbox

Details of the formulation:

A variable for each node, a constraint for each arc



variables:

```
at_b_h_0, at_c_h_0, at_j_h_0,
at_b_h_0, load_j_c_h_0, load_b_c_h_0
...
```

constraints:

1. **not** load_j_c_h_0 **or** at_j_h_0
2. **not** load_j_c_h_0 **or** at_c_h_0
3. **not** load_j_c_h_0 **or** in_j_c_1
4. **not** load_j_c_h_0 **or not** at_j_c_1

Always: action => literal

Blackbox: algorithm

```
function BLACKBOX(problem):
         graph = GRAFO_INIZIALE (problem)
         targets = GOAL (problem)
         do loop:
            if objectives not mutex in last step:
               convert the graph into a SAT problem
               Sol = SEARCH (SAT problem, solver)
               if Sol ≠ fail: return Sol
               else if LEVEL_OFF (graph): return fail
            = ESPANDI_GRAFO graph (graph, problem)
```

- With respect to graphplan we only change the way of doing search (and the encoding, of course)
- You can use any SAT solver, or even the same graphplan

Exercise

Start blackbox

- Blackbox is already installed on the lab machines
- Try to run it with:

```
gigi@lab2:~$ blackbox
```

- Downloaded from the course website the archive "cart_example.zip"
- Run the program using as input the two .pddl and .facts files that you have found in the "Cart Example":

```
blackbox -o <.pddl file> -f <.facts file>
```

Exercise

Try to …

1. Understand which method blackbox is using to find a valid plan (maybe try to vary the output level).
2. Analyse the WFF (Well Formed Formula) to which the planning graph to the last stage is converted: try to understand it at least a little bit.
3. Employ blackbox by using different solvers for finding a valid plan (the online help does not say that, but you can also use CHAFF).

PDDL

Planning Domain Definition Language

Some nuts on PDDL

Blackbox and FF require that the files of the facts and actions are in PDDL.

PDDL = Planning Domain Definition Language
Attempt to standardizs a language for modeling planning problems

- Developed to support the International Planning Competition

A PDDL definition consists of two parts

- A domain (operators + objects)
- A problem

Typically in separate files

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

characteristics of the language actually used

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

Object types that are part of the domain

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

Types of predicate; eg at (home, car)

PDDL - Domain

Approximate syntax of a domain definition

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

operators
Description

PDDL - predicates and actions

Syntax for variables with type:

```
?<Var name> - <type>
```

Syntax of a formula

Aggregation of predicates using logical operators:

```
(and PREDICATE_1 ... PREDICATE_N)
(or PREDICATE_1 ... PREDICATE_N)
(not PREDICATE)
```

Effects

Add and delete lists are aggregated (the delete effect are preceded by a "not" :

```
:effect FORMULA
```

PDDL - .facts File

```
(define (<problem name>)
        (:domain <reference domain>)
        (:objects
                {<object name> - <type>}
        )
        (:init
                {<predicate in the initial state>}
        )
        (:goal
                (and {<predicate in the final state>})
        )
)
```

Input of a planner = two files:

- Description of actions and types of objects (domain, .pddl file)
- Description of the initial state and the goal (.facts file)

References

- **GRAPHPLAN**
  - http://www.cs.cmu.edu/~avrim/graphplan.html
  - A. M. Blum and Furst, "Fast Planning Planning Through Graph Analysis", Artificial Intelligence, 90: 281-300 (1997).
- **Blackbox**
  - http://www.cs.rochester.edu/u/kautz/satplan/blackbox/index.html
  - SATPLAN: http://www.cs.rochester.edu/u/kautz/satplan/index.htm
  - Henry Kautz and Bart Selman, "Unifying SAT-based and Graph-based Planning", Proc. IJCAI-99, Stockholm, 1999.
- **Fast Forward**
  - http://members.deri.at/~joergh/ff.html
  - J. Hoffmann, "FF: The Fast-Forward Planning System", in: AI Magazine, Volume 22, Number 3, 2001, Pages 57-62

All items are available on the course website

Exercises

graph-based planning (& Friends)

Exercise 1

Modeling in the PDDL the **nine puzzle problem**

`initial state`

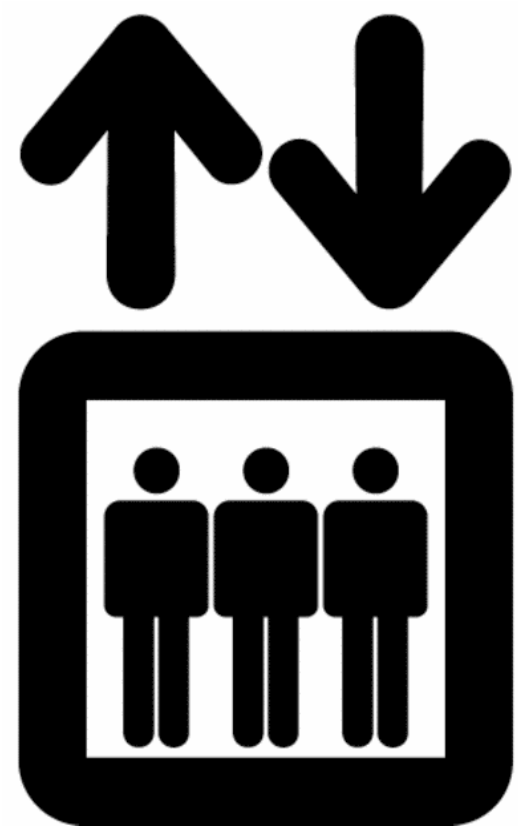
7	3	4
2		1
8	6	5

`Goal`

1	2	3
4	5	6
7	8	

- Try to solve it with the various planners
- How to vary performances by "mixing" a little '?

Exercise 2



- A building has 5 floors, 5 tenants and two lifts (currently at the ground floor and at the 4th floor)
- Each tenant is at a starting floor and must go to another floor
- We need to find a plan that moves each tenant to its destination and that requires the minimum possible time (Load / unload the lift takes the same time to move up one level).

Model the problem in PDDL and solve it!!

Exercise 2: Tenants

the young **Enrico** is at the ground floor after a hard morning at work and is finally back home on the 3rd floor

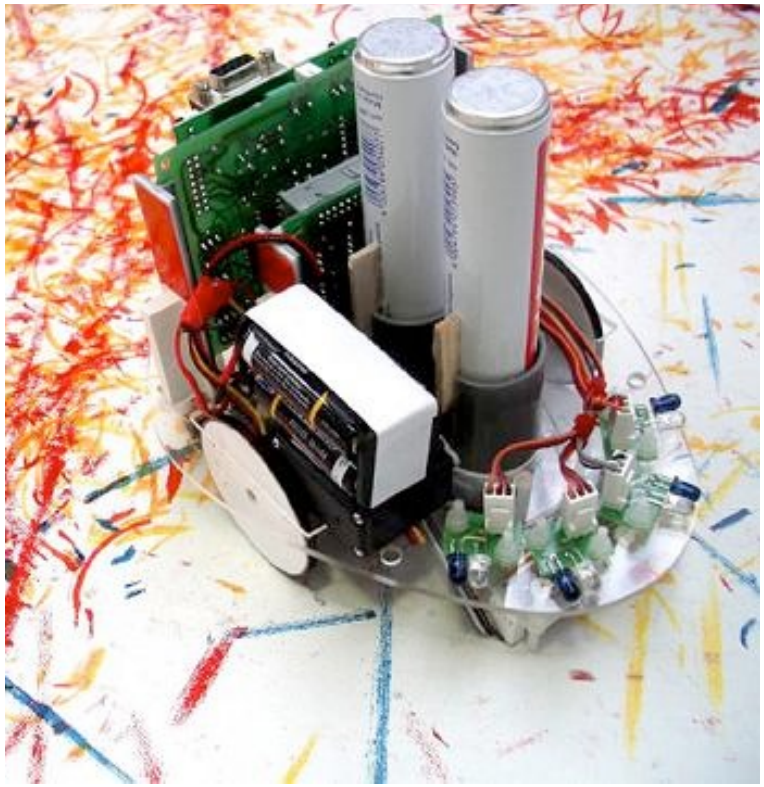
After a family lunch on the 3rd floor, **Ettore** should go back to his workshop on the ground floor

Elena is at the ground floor should bring her shopping bags on the 2nd floor

Elvira and **Ennio** are on the 1st (to play cards) and on the 2nd (composing music) and should go back home on the 4th floor

Exercise 3

Two robots have to paint the tiles on a floor:



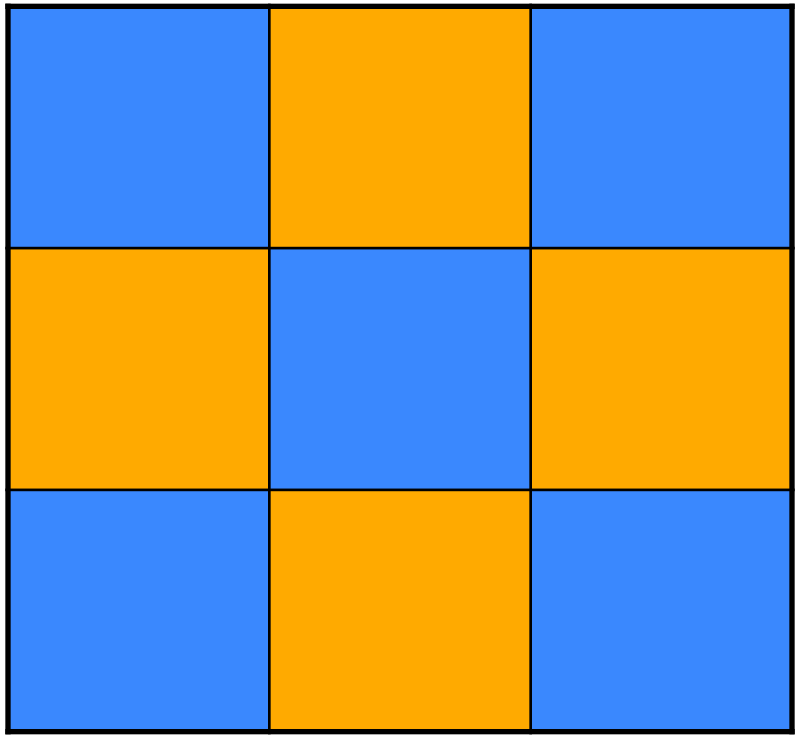
- Each robot can move between adjacent tiles
- Each robot is loaded with a specific color and can use the color on the tile it is located to.
- Robots cannot pass on a painted tile
- The two robots can be on the same tile

Exercise 3

initial state

BLUE		
		YELLOW

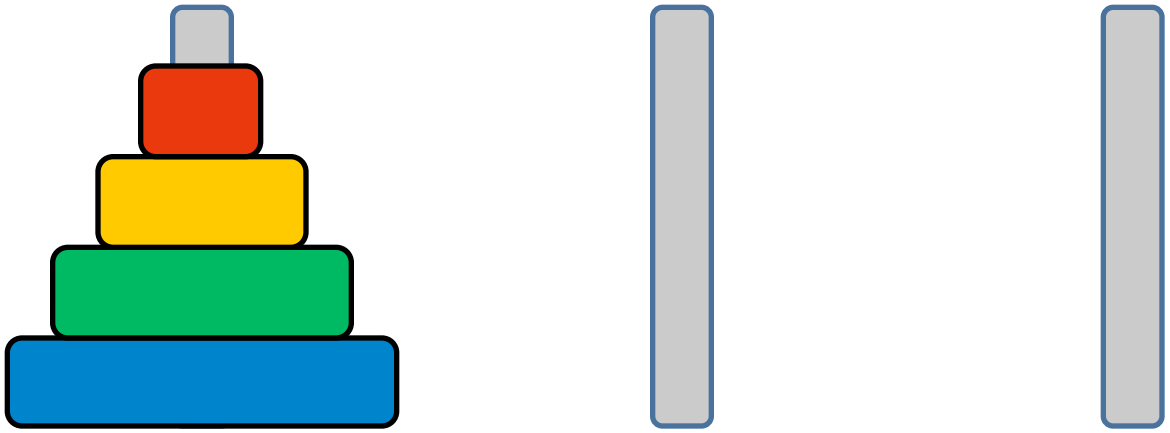
Goal



Note that a robot can color a tile, even if the other robot there is located above at the time

Exercise 4

Shape in the puzzle of PDDL **Hanoi Tower**



- Move a circle at a time, from one busbar
- A smaller circle can never go above a larger one

Exercise 4

Goal: