Solving a PPDDL Problem as a Simple Stochastic Game

Lorenzo Mandelli (1747091)
Nicolò Paoletti (1888157)
Valerio Ponzi (1760886)
Giulio Recupito (1668320)

SAPIENZA
UNIVERSITÀ DI ROMA

Summary:

- PDDL
- PPDDL
- GRAPH CREATION
- SOLVER (Simplex method)
- TIME COMPLEXITY ANALYSIS
- CONCLUSIONS

PDDL (Planning Domain Definition Language)

- Standard encoding language for “classical” planning tasks
- Components of a PDDL planning task:
  - Objects: Things in the world that interest us.
  - Predicates: Properties of these objects (true or false).
  - Initial state: The state of the world that we start in.
  - Goal specification: Things that we want to be true.
  - Actions: Ways of changing the state of the world.

PDDL (Planning Domain Definition Language)

Two files:

1. The Domain file for predicates and actions.
2. A Problem file for object, initial state and goal.

SAPIENZA
UNIVERSITÀ DI ROMA

Problem example:

```
(define (problem test-problem)
(:domain test-domain)
(:objects ups-box cereal-box - box)
(:init (parked goldie) (holding ups-box) (in cereal-box goldie) (= (fuel-level goldie) 7))
(:goal (and (in ups-box goldie) (>= (fuel-level goldie) 9))))
```

Domain example:

```
(define (domain test-domain)
(:requirements :typing :equality
:conditional-effects :fluents)
(:types car box)
(:constants goldie - car)
(:predicates (parked ?x - car) (holding ?x - box)
(in ?x - box ?y - car))
(:functions (fuel-level ?x - car))
```

Action

```
(:action load
    :parameters (?x - box ?y - car)
    :precondition (and (holding ?x) (parked ?y))
    :effect (and (in ?x ?y)
        (forall (?z - car)
            (when (not (= ?z ?y))
                (not (in ?x ?z))))))
```

Action

```
(:action refuel
    :parameters (?x - car)
    :precondition (< (fuel-level ?x) 10)
    :effect (increase (fuel-level ?x) 1)))
```

PPDDL (Probabilistic + Planning Domain Definition Language)

This extension give us a general language for describing probabilistic and decision theoretic planning problems !

The syntax for probabilistic effects is:

(probabilistic $p_1 e_1 \dots p_k e_k$) *with p_i probability of effect i*

We required that $p_i \geq 0$ and $\sum^{k-1}_{i=1} p_i \leq 1$

SAPIENZA
UNIVERSITÀ DI ROMA

PPDDL example

Action

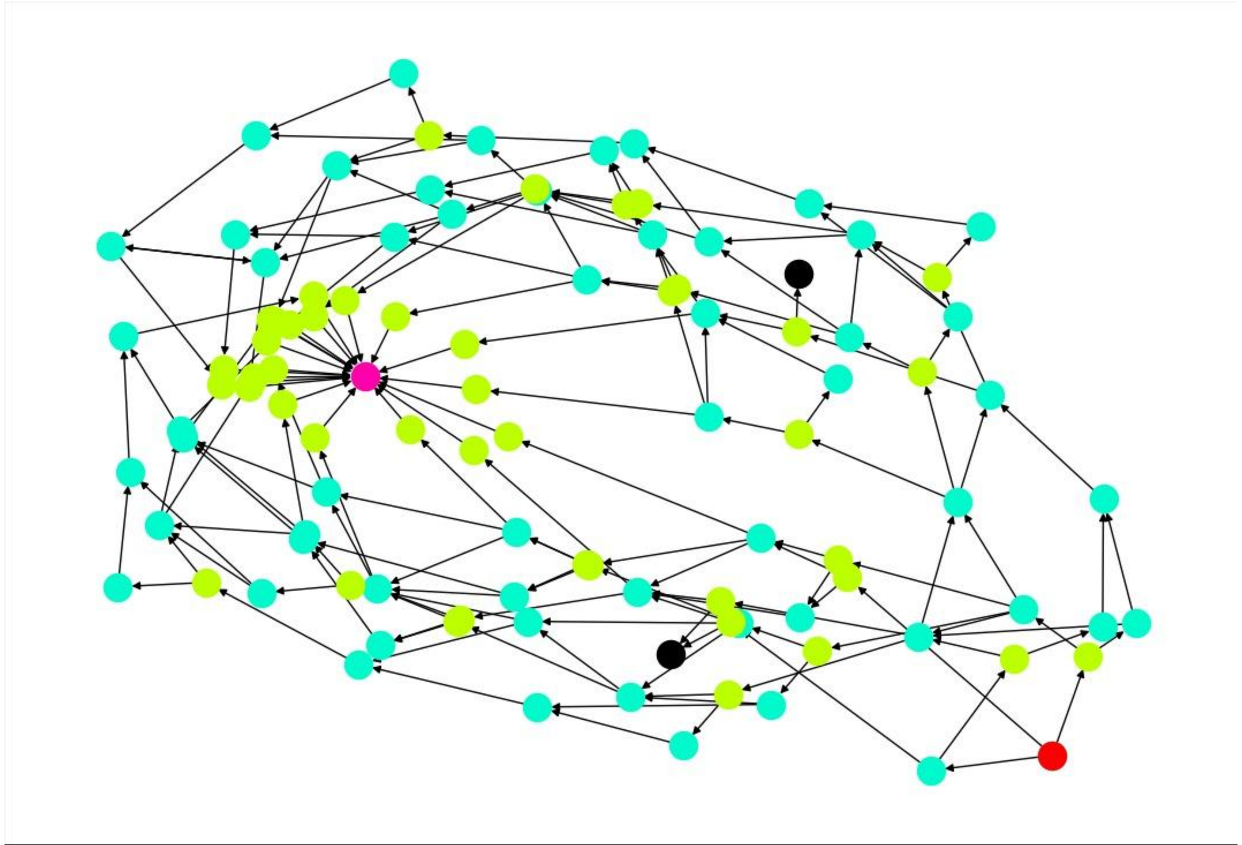
```
(define (domain bomb-and-toilet)
(:requirements :conditional-effects :probabilistic-effects)
(:predicates (bomb-in-package ?pkg) (toilet-clogged) (bomb-defused))
(:action dunk-package
    :parameters (?pkg)
    :effect (and (when (bomb-in-package ?pkg) (bomb-defused))
            (probabilistic 0.05 (toilet-clogged)))))

(define (problem bomb-and-toilet)
    (:domain bomb-and-toilet)
    (:requirements :negative-preconditions)
    (:objects package1 package2)
    (:init (probabilistic 0.5 (bomb-in-package package1) 0.5
    (bomb-in-package package2)))
    (:goal (and (bomb-defused) (not (toilet-clogged)))))
```

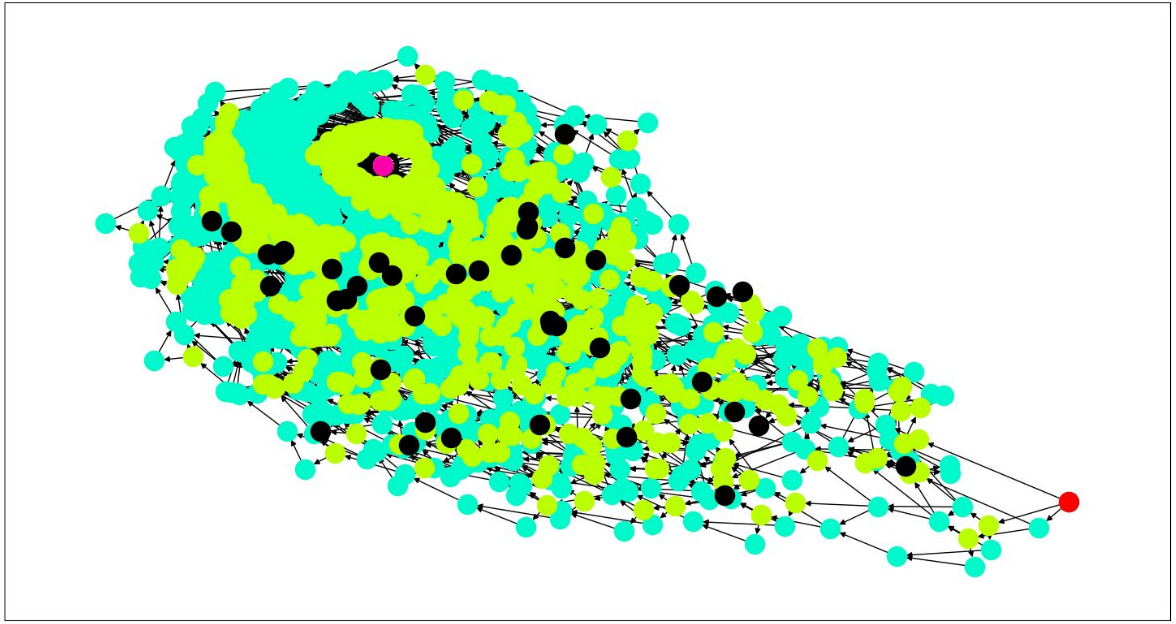
Graph

- We represented a common PPDDL program as a directed graph
- Start node is the root node with no incoming edges
- Goal node is the end node with no outcoming edges
- Max nodes contain information about the state
- Average nodes are like gates where the edges branch, they do no contains information about the state
- Sink nodes are nodes different from the goal node, still without outcoming edges
- Edges contain information about the action taking place in the origin node and the probability that action takes place

P01 Graph



P02 Graph



SSG

- A simple stochastic game is a direct graph G = (V,E)
- The vertex set V is the union of disjoint sets V_max , V_avg , together with two special vertices, the 0-sink and the 1-sink

Given the graph representation we implemented, **we can now treat a PPDDL problem as a SSG**

Solver

A **strategy τ** in this scenario is than a set of edges of E, each with its left hand at a max vertex, such that for each max vertex i there is exactly one edge (i,j) in τ

The solution of G = (V,E) be an SSG with n vertices, is the **optimal solution of a simple constrained minimization linear programming problem**

We can move the task to a simple optimization problem, solvable by means of the **Simplex Method.**

Simplex Method

The simplex algorithm operates on linear programs in the **canonical form (1)**, or in its equivalent **tableau form (2)**

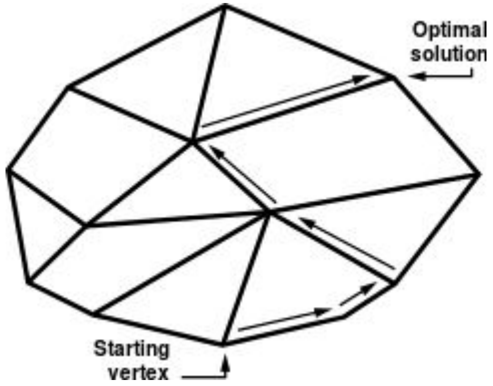
$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b, \\ & x \geq 0 \end{aligned} \quad (1)$$

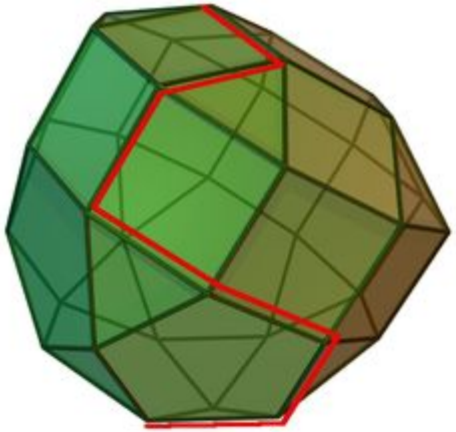
$$\begin{bmatrix} 1 & -c^T & 0 \\ 0 & A & b \end{bmatrix} \quad (2)$$

- $c = (c_1, \ldots, c_n)$ the coefficients of the objective function
- $x = (x_1, \ldots, x_n)$ the variable of the problem
- A a pxn matrix
- $b = (b_1 , \ldots ,b_p)$ the known terms.

Simplex Method: Geometrical Interpretation

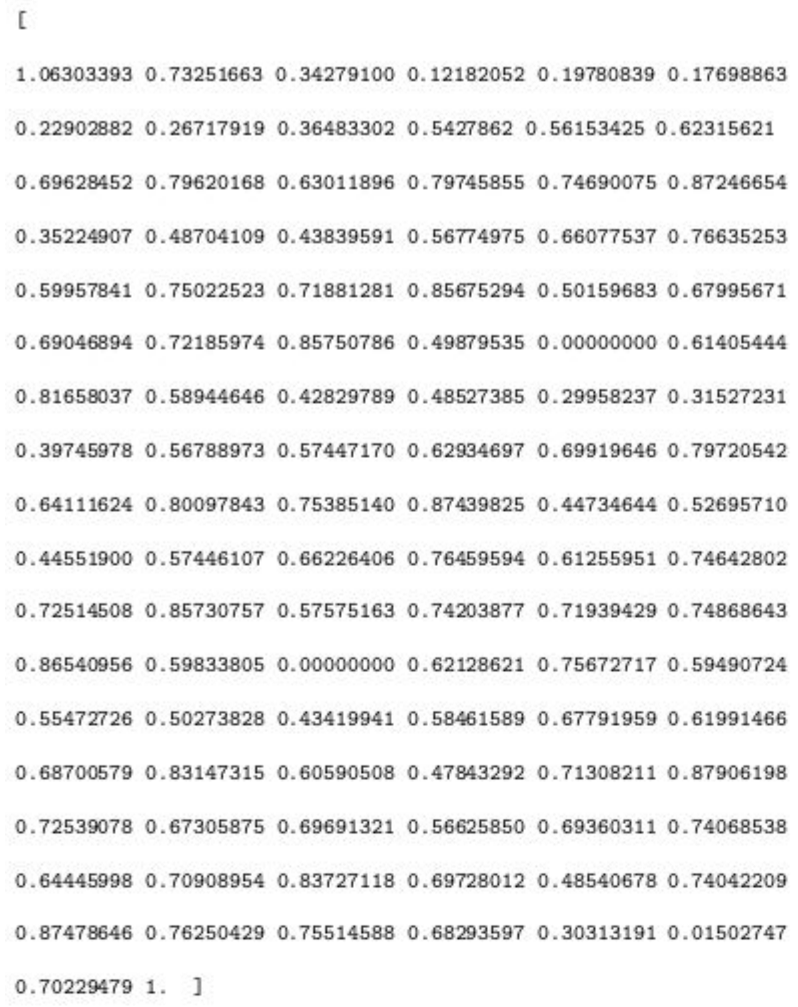
In geometric terms, the feasible region defined by all the values of x such that $Ax \leq b$ and $\forall\ i,\ x\_i \geq 0$ is a (possibly unbounded) **convex polytope**





Simplex Method: Output

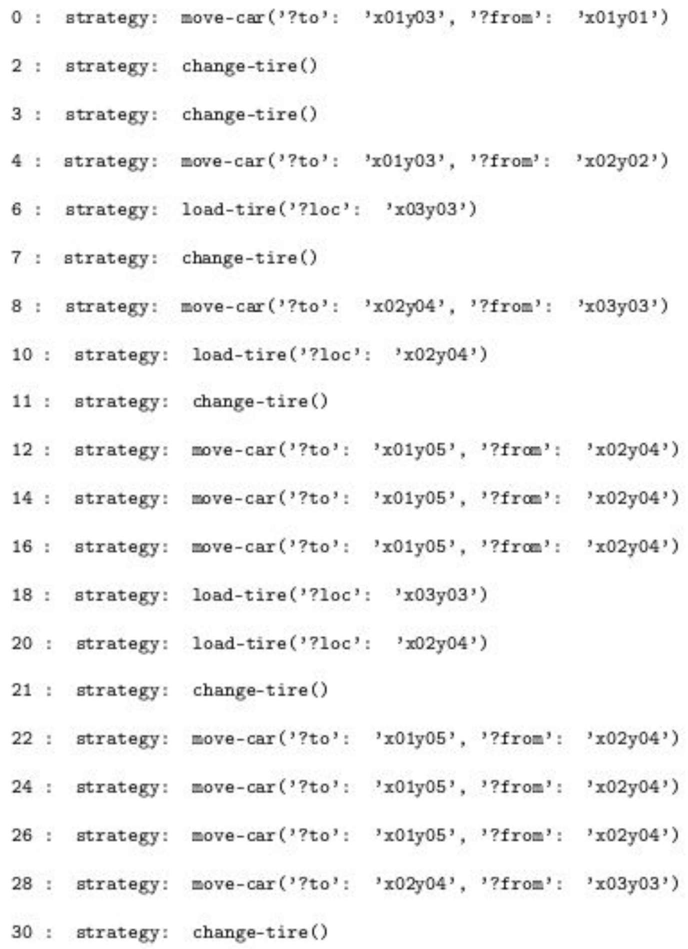
The simplex method output an **array of n elements**, one for each node of the graph, **containing the probability that starting from that node, we end up in the goal node**.



```
[
1.06303393 0.73251663 0.34279100 0.12182052 0.19780839 0.17698863
0.22902882 0.26717919 0.36483302 0.5427862 0.56153425 0.62315621
0.69628452 0.79620168 0.63011896 0.79745855 0.74690075 0.87246654
0.35224907 0.48704109 0.43839591 0.56774975 0.66077537 0.76635253
0.59957841 0.75022523 0.71881281 0.85675294 0.50159683 0.67995671
0.69046894 0.72185974 0.85750786 0.49879535 0.00000000 0.61405444
0.81658037 0.58944646 0.42829789 0.48527385 0.29958237 0.31527231
0.39745978 0.56788973 0.57447170 0.62934697 0.69919646 0.79720542
0.64111624 0.80097843 0.75385140 0.87439825 0.44734644 0.52695710
0.44551900 0.57446107 0.66226406 0.76459594 0.61255951 0.74642802
0.72514508 0.85730757 0.57575163 0.74203877 0.71939429 0.74868643
0.86540956 0.59833805 0.00000000 0.62128621 0.75672717 0.59490724
0.55472726 0.50273828 0.43419941 0.58461589 0.67791959 0.61991466
0.68700579 0.83147315 0.60590508 0.47843292 0.71308211 0.87906198
0.72539078 0.67305875 0.69691321 0.56625850 0.69360311 0.74068538
0.64445998 0.70908954 0.83727118 0.69728012 0.48540678 0.74042209
0.87478646 0.76250429 0.75514588 0.68293597 0.30313191 0.01502747
0.70229479 1.  ]
```

Strategy

The strategy is defined in a greedy way: **for each max node**, the only we can control, **we check all its neighbours, and select the one maximizing the probability of ending up in the goal node.**



```
0 :  strategy:  move-car('?to':  'x01y03', '?from':  'x01y01')
2 :  strategy:  change-tire()
3 :  strategy:  change-tire()
4 :  strategy:  move-car('?to':  'x01y03', '?from':  'x02y02')
6 :  strategy:  load-tire('?loc':  'x03y03')
7 :  strategy:  change-tire()
8 :  strategy:  move-car('?to':  'x02y04', '?from':  'x03y03')
10 :  strategy:  load-tire('?loc':  'x02y04')
11 :  strategy:  change-tire()
12 :  strategy:  move-car('?to':  'x01y05', '?from':  'x02y04')
14 :  strategy:  move-car('?to':  'x01y05', '?from':  'x02y04')
16 :  strategy:  move-car('?to':  'x01y05', '?from':  'x02y04')
18 :  strategy:  load-tire('?loc':  'x03y03')
20 :  strategy:  load-tire('?loc':  'x02y04')
21 :  strategy:  change-tire()
22 :  strategy:  move-car('?to':  'x01y05', '?from':  'x02y04')
24 :  strategy:  move-car('?to':  'x01y05', '?from':  'x02y04')
26 :  strategy:  move-car('?to':  'x01y05', '?from':  'x02y04')
28 :  strategy:  move-car('?to':  'x02y04', '?from':  'x03y03')
30 :  strategy:  change-tire()
```

Complexity

In order to analyze the computational complexity of our solution, we have to take into account:

- The computational complexity of the graph creation
- The computational complexity of the planner

Graph complexity

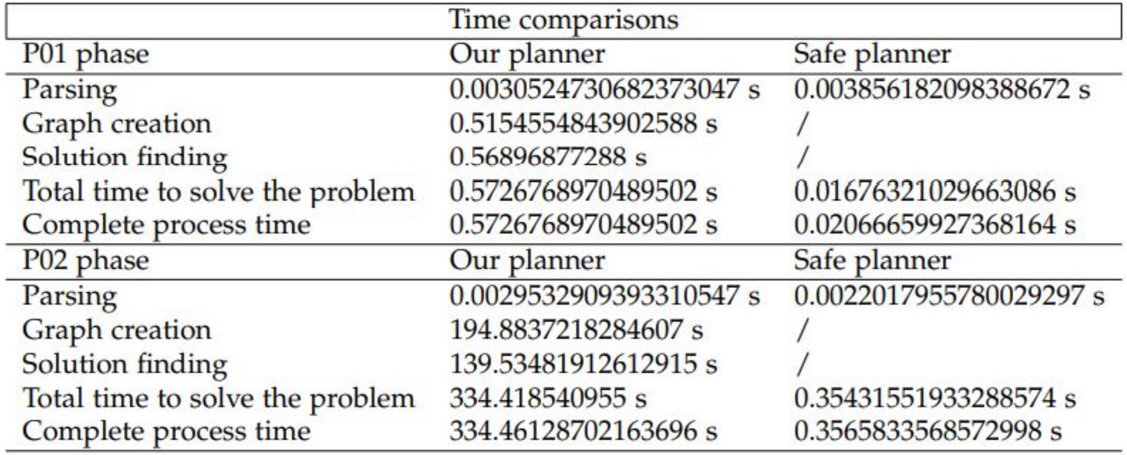
- The computational complexity is $O(|A|^n)$

  where $|A|$ is the set of actions and n is the maximum distance (depth) from the starting node.

- The resulting time complexity is exponential due to the creation of the graph

Conclusion

We compared the time results with a very recent planner called Safe-Planner, using two different instances: the triangle-tireworld problem P01 and P02.



Time comparisons		
P01 phase	Our planner	Safe planner
Parsing	0.0030524730682373047 s	0.003856182098388672 s
Graph creation	0.5154554843902588 s	/
Solution finding	0.56896877288 s	/
Total time to solve the problem	0.5726768970489502 s	0.01676321029663086 s
Complete process time	0.5726768970489502 s	0.02066659927368164 s
P02 phase	Our planner	Safe planner
Parsing	0.0029532909393310547 s	0.0022017955780029297 s
Graph creation	194.8837218284607 s	/
Solution finding	139.53481912612915 s	/
Total time to solve the problem	334.418540955 s	0.35431551933288574 s
Complete process time	334.46128702163696 s	0.3565833568572998 s

Table 1: Comparison between our planner and SP execution time.

Final remarks

- The safe planner is exponentially faster in finding a solution, since planner algorithms used do not require to expand and create all the search space exploiting various heuristics.

- Unlike our solution need this procedure in order to create the graph. The time to find the solution is then affected, since the simplex has to work with a number of node which is exponential in the depth of the solution.