Web-Planner

A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search

Maurício C. Magnaguagno, Ramon Fraga Pereira, Martin D. Móre, Felipe Meneguzzi
School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre - RS, Brazil
{mauricio.magnaguagno, ramon.pereira, martin.more}@acad.pucrs.br, felipe.meneguzzi@pucrs.br

Introduction

- **Classical planning**
  - Declarative domain specification
  - Opaque intermediary steps
  - Challenging task for new users
  - Fixing mistakes is non-trivial

- **Heuristic Functions**
  - Modern classical planners
  - Different domains ⇒ different heuristic functions
  - Evaluate and select the best heuristic function

Introduction

- **Planners have no easy setup**
  - Academic projects
  - Small to no documentation
- **No extra information**
  - Planning failure gives no hint to the user
  - Is it impossible or incorrectly described?
  - How far the planner got until something went wrong?

- **Solution**
  - Move planner to the cloud (no setup)
  - Visualize internal data structures (explore)

Background - Classical Planning

Domain

- How the world "works"
- Predicates ⇒ Features
- Actions ⇒ Transitions
  - Preconditions
  - Effects

- Does the domain match the real world?

Problem

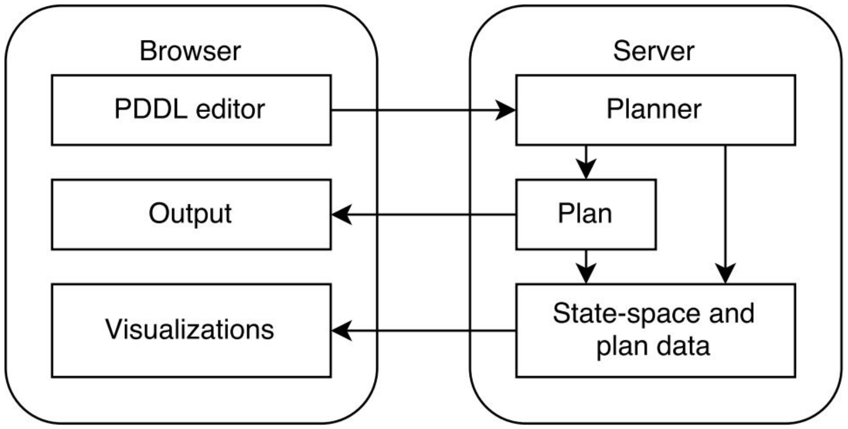
- How the world is now
- Objects
- Initial state
- Goal state

- Is there a plan that reaches the goal?

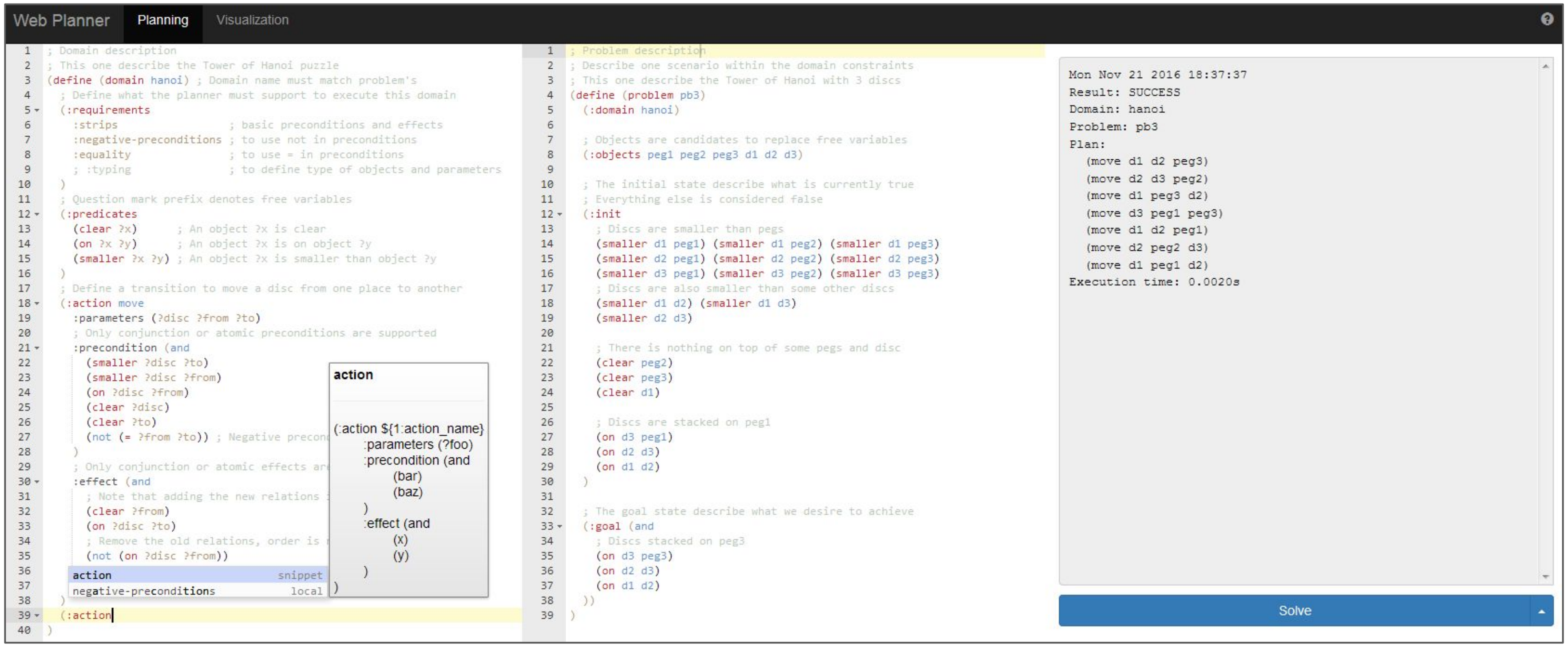
Web Planner Architecture

- Interactions in the user-side
- Planning and data gathering in the server-side

- JSON as intermediate representation



Domain Development Interface



```
; Domain description
; This one describe the Tower of Hanoi puzzle
(define (domain hanoi) ; Domain name must match problem's
  ; Define what the planner must support to execute this domain
  (:requirements
    :strips                 ; basic preconditions and effects
    :negative-preconditions ; to use not in preconditions
    :equality               ; to use = in preconditions
    ; :typing               ; to define type of objects and parameters
  )
  ; Question mark prefix denotes free variables
  (:predicates
    (clear ?x)      ; An object ?x is clear
    (on ?x ?y)      ; An object ?x is on object ?y
    (smaller ?x ?y) ; An object ?x is smaller than object ?y
  )
  ; Define a transition to move a disc from one place to another
  (:action move
    :parameters (?disc ?from ?to)
    ; Only conjunction or atomic preconditions are supported
    :precondition (and
      (smaller ?disc ?to)
      (smaller ?disc ?from)
      (on ?disc ?from)
      (clear ?disc)
      (clear ?to)
      (not (= ?from ?to)) ; Negative precon
    )
    ; Only conjunction or atomic effects ar
    :effect (and
      ; Note that adding the new relations i
      (clear ?from)
      (on ?disc ?to)
      ; Remove the old relations, order is 
      (not (on ?disc ?from))
  )
  (:action
)
```

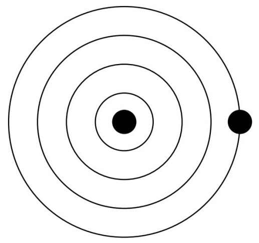
```
action                         snippet
negative-preconditions         local
```

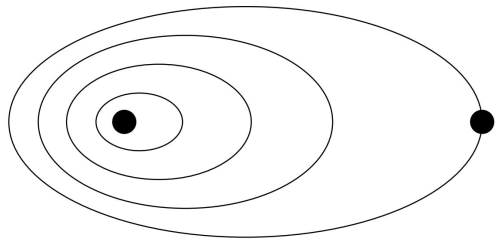
```
action

(:action ${1:action_name}
    :parameters (?foo)
    :precondition (and
        (bar)
        (baz)
    )
    :effect (and
        (x)
        (y)
    )
)
```

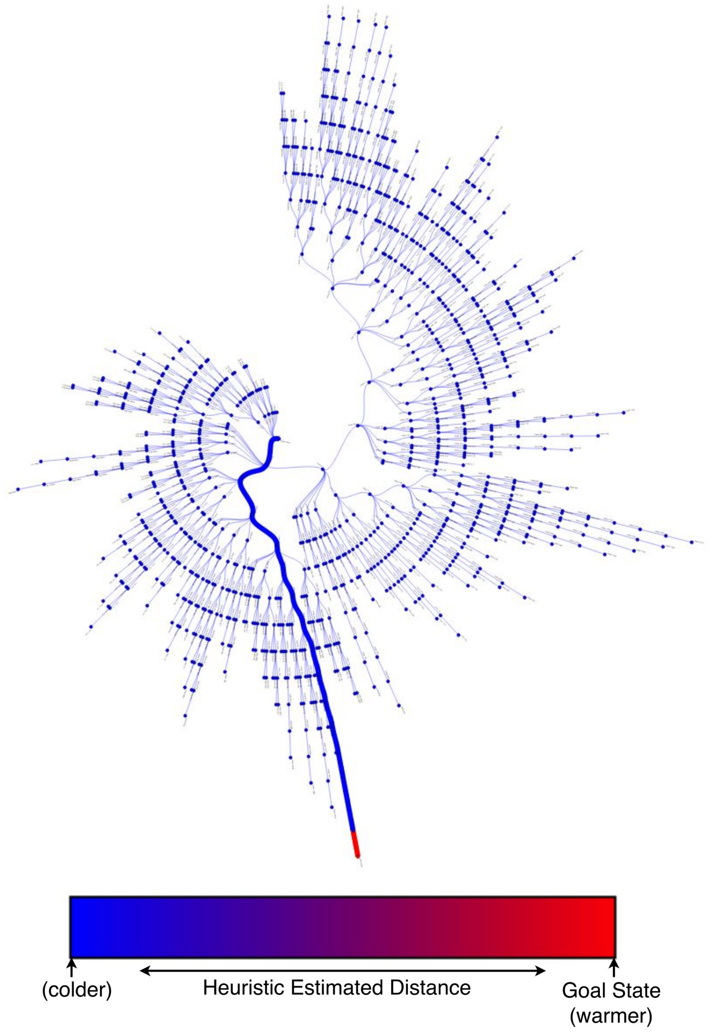
```
; Problem description
; Describe one scenario within the domain constraints
; This one describe the Tower of Hanoi with 3 discs
(define (problem pb3)
  (:domain hanoi)

  ; Objects are candidates to replace free variables
  (:objects peg1 peg2 peg3 d1 d2 d3)

  ; The initial state describe what is currently true
  ; Everything else is considered false
  (:init
    ; Discs are smaller than pegs
    (smaller d1 peg1) (smaller d1 peg2) (smaller d1 peg3)
    (smaller d2 peg1) (smaller d2 peg2) (smaller d2 peg3)
    (smaller d3 peg1) (smaller d3 peg2) (smaller d3 peg3)
    ; Discs are also smaller than some other discs
    (smaller d1 d2) (smaller d1 d3)
    (smaller d2 d3)

    ; There is nothing on top of some pegs and disc
    (clear peg2)
    (clear peg3)
    (clear d1)

    ; Discs are stacked on peg1
    (on d3 peg1)
    (on d2 d3)
    (on d1 d2)
  )

  ; The goal state describe what we desire to achieve
  (:goal (and
    ; Discs stacked on peg3
    (on d3 peg3)
    (on d2 d3)
    (on d1 d2)
  ))
)
```

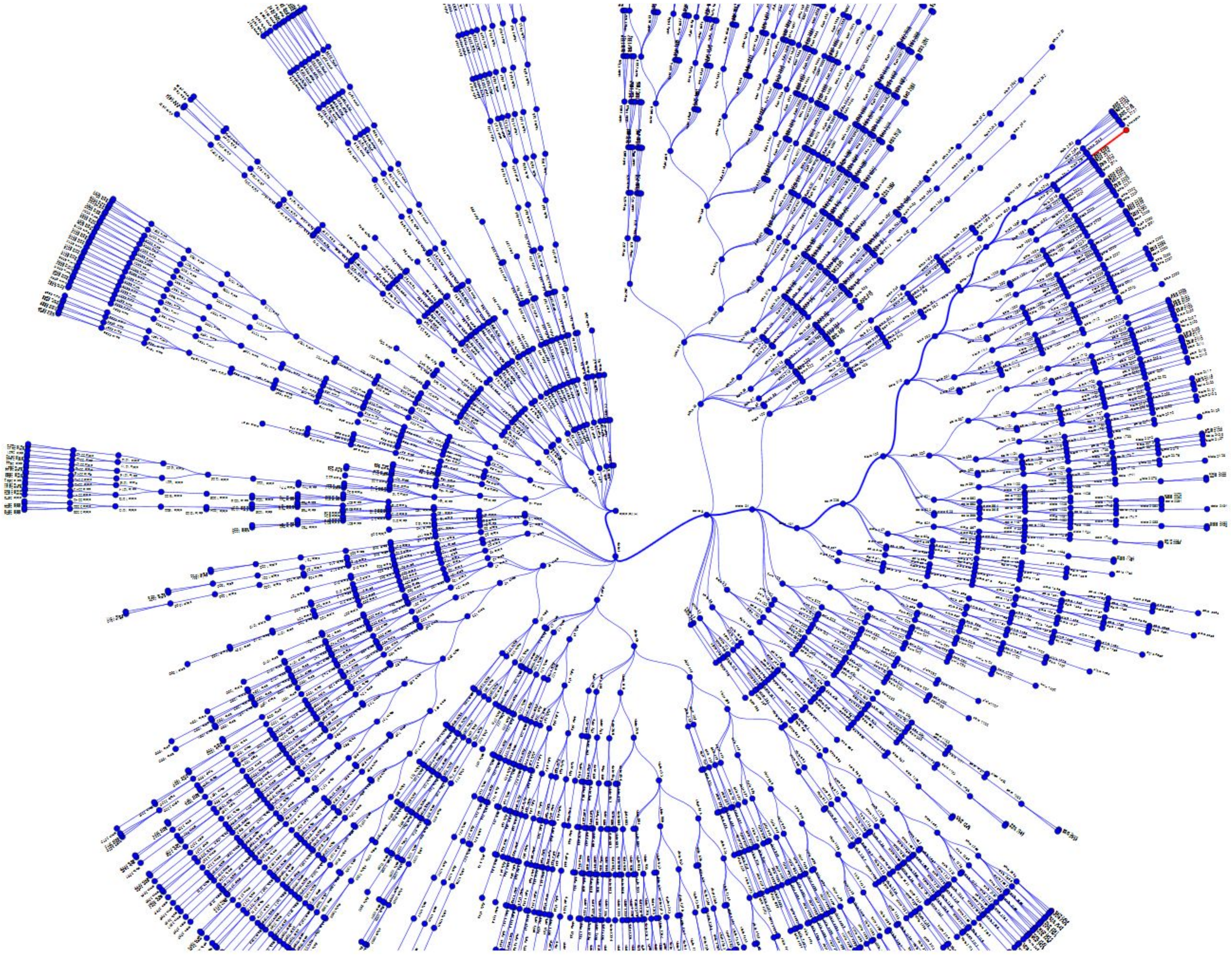
```
Mon Nov 21 2016 18:37:37
Result: SUCCESS
Domain: hanoi
Problem: pb3
Plan:
   (move d1 d2 peg3)
   (move d2 d3 peg2)
   (move d1 peg3 d2)
   (move d3 peg1 peg3)
   (move d1 d2 peg1)
   (move d2 peg2 d3)
   (move d1 peg1 d2)
Execution time: 0.0020s
```

Visualization Interface - Search

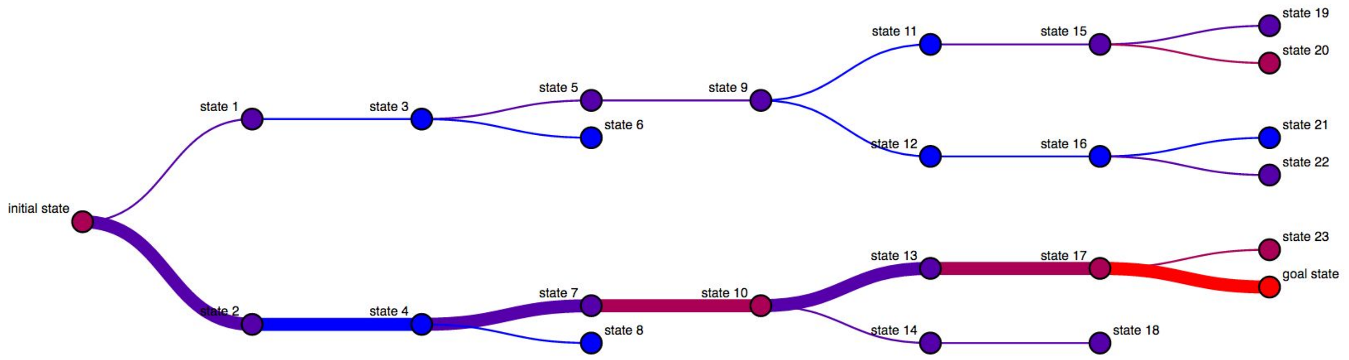


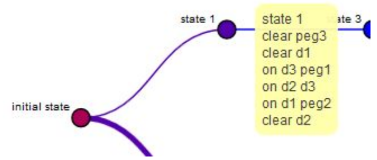




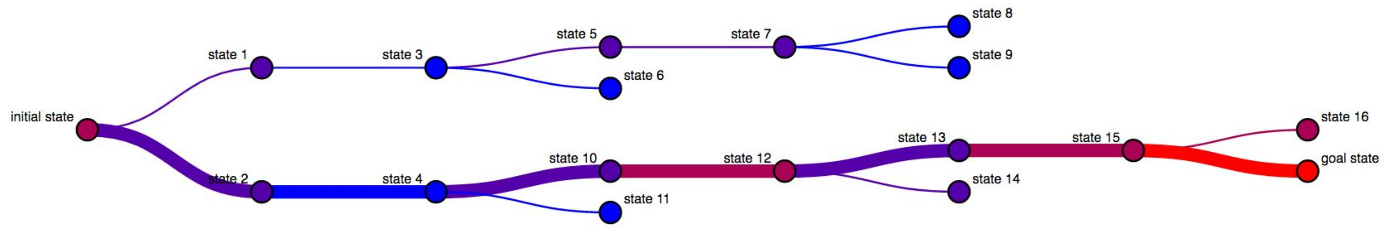


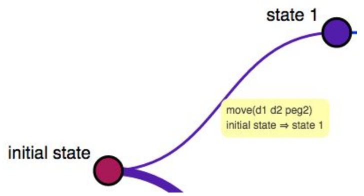
Visualization Interface - Search





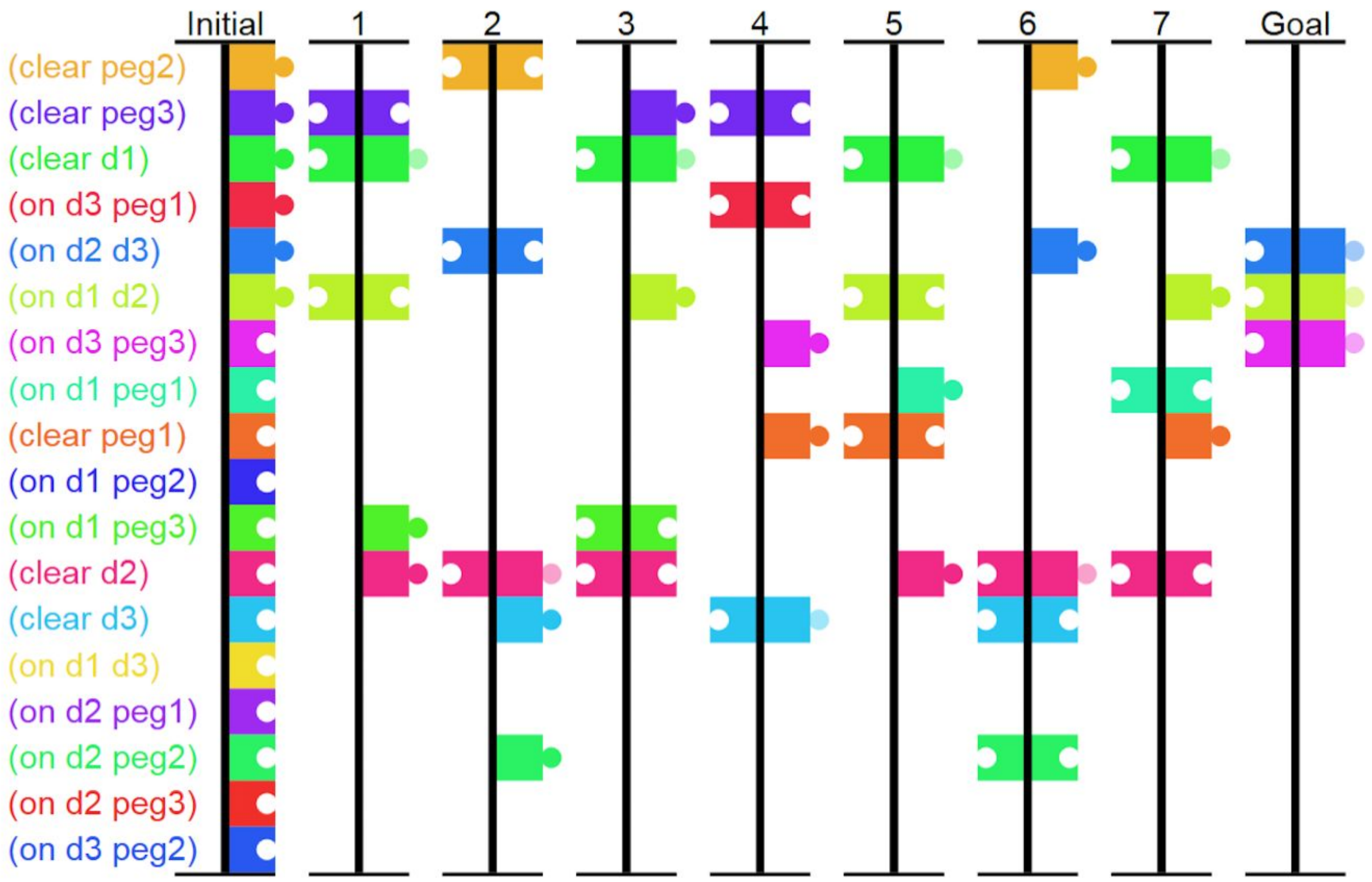
(1) Breadth First Search





(2) Best First Search with Hamming distance

Visualization Interface - Plan

- 

Survey Results

The survey contained the following questions and answers (5 users):

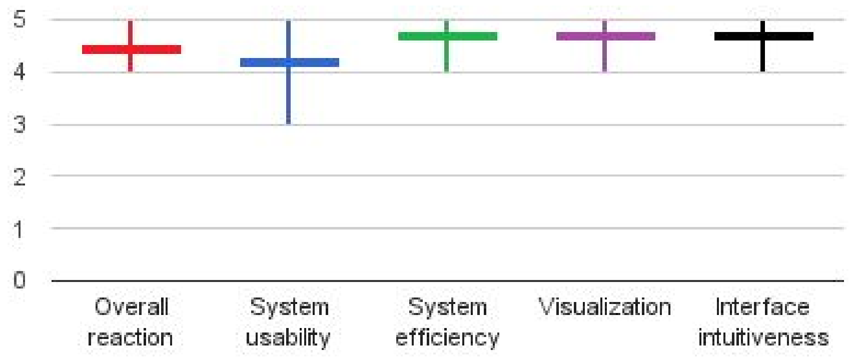
How familiar are you with automated planning languages and algorithms?

- Have used PDDL before (1)

Did the visualizations help you to find any bugs/errors/interesting points during the course of your task?

- Found missing preconditions (1)

System Reaction



Survey Results

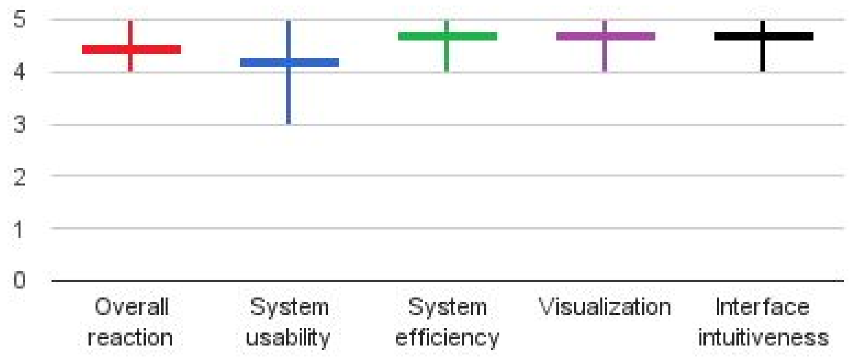
Mark other planners/tools you used in your experiments:

- Fast-Downard (1)
- JavaFF (1)
- JavaGP (3)
- Planning.domains (3)
- STRIPS-Fiddle (1)

Which features you missed the most?

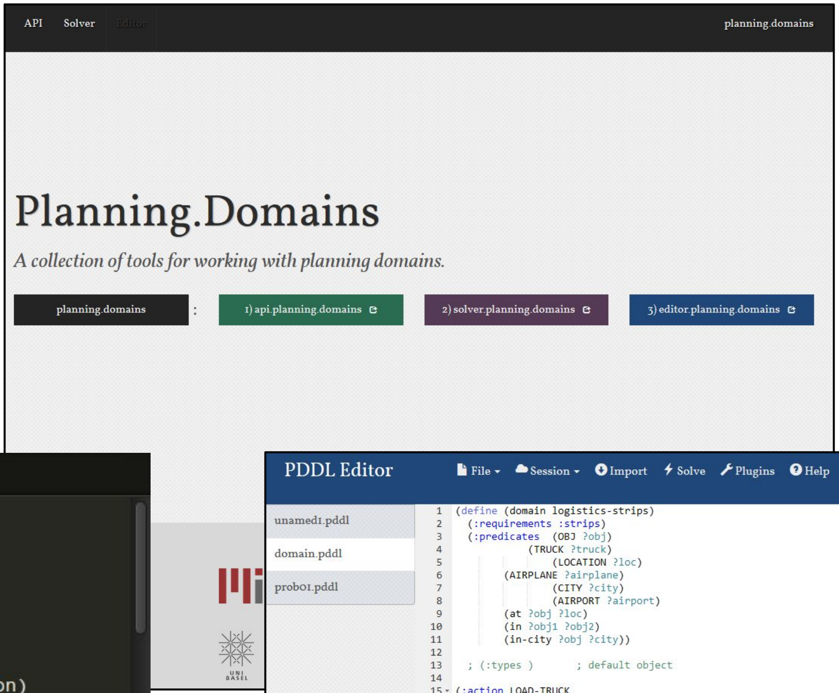
- Support more requirements (2)
- Auto-complete (1)
- Option to clear console (1)
- Find (common) errors in PDDL (1)

System Reaction



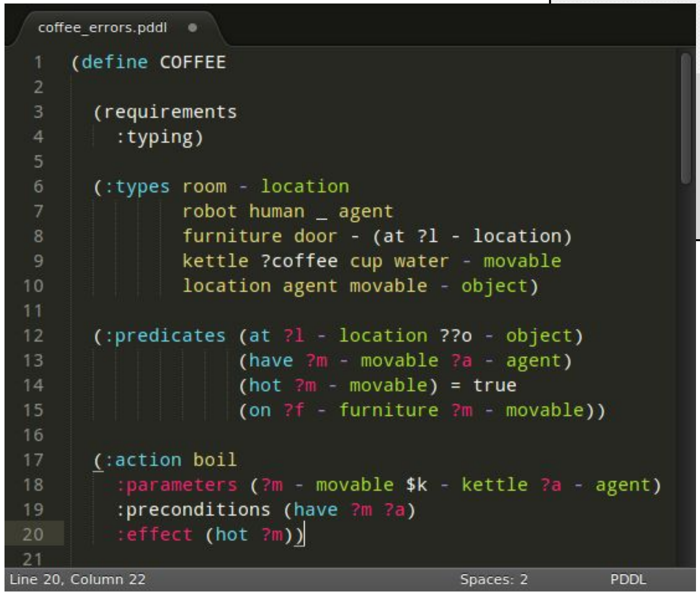
Related Work

- Planning.domains
- myPDDL
- ...

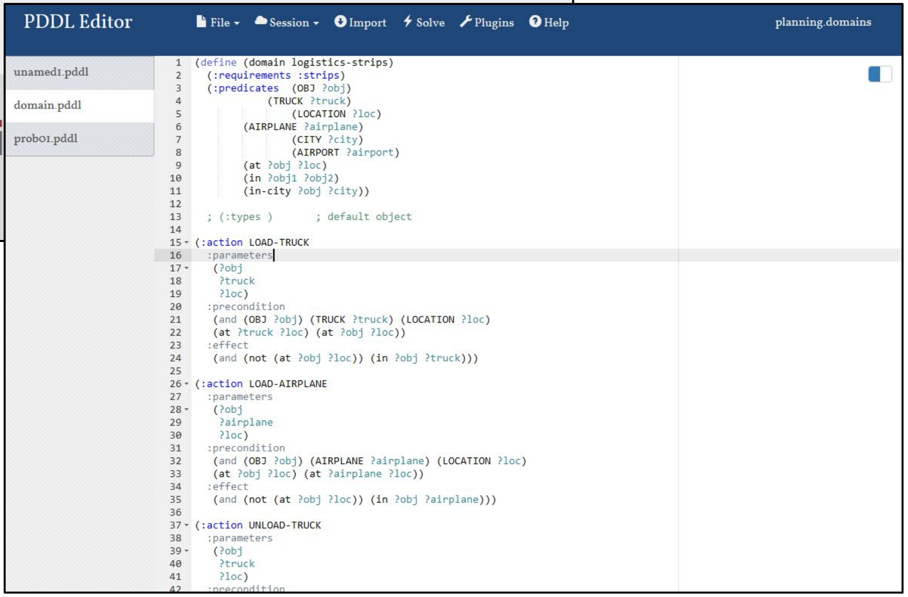


Planning.Domains

A collection of tools for working with planning domains.



```
(define COFFEE

  (requirements
    :typing)

  (:types room - location
          robot human _ agent
          furniture door - (at ?l - location)
          kettle ?coffee cup water - movable
          location agent movable - object)

  (:predicates (at ?l - location ??o - object)
               (have ?m - movable ?a - agent)
               (hot ?m - movable) = true
               (on ?f - furniture ?m - movable))

  (:action boil
    :parameters (?m - movable $k - kettle ?a - agent)
    :preconditions (have ?m ?a)
    :effect (hot ?m))
```



```
(define (domain logistics-strips)
  (:requirements :strips)
  (:predicates  (OBJ ?obj)
            (TRUCK ?truck)
                (LOCATION ?loc)
        (AIRPLANE ?airplane)
                (CITY ?city)
                (AIRPORT ?airport)
        (at ?obj ?loc)
        (in ?obj1 ?obj2)
        (in-city ?obj ?city))

  ; (:types )       ; default object

(:action LOAD-TRUCK
  :parameters
   (?obj
    ?truck
    ?loc)
  :precondition
   (and (OBJ ?obj) (TRUCK ?truck) (LOCATION ?loc)
   (at ?truck ?loc) (at ?obj ?loc))
  :effect
   (and (not (at ?obj ?loc)) (in ?obj ?truck)))

(:action LOAD-AIRPLANE
  :parameters
   (?obj
    ?airplane
    ?loc)
  :precondition
   (and (OBJ ?obj) (AIRPLANE ?airplane) (LOCATION ?loc)
   (at ?obj ?loc) (at ?airplane ?loc))
  :effect
   (and (not (at ?obj ?loc)) (in ?obj ?airplane)))

(:action UNLOAD-TRUCK
  :parameters
   (?obj
    ?truck
    ?loc)
  :precondition
```

Conclusions and Future Work

- Make planning easier to setup
  - PDDL editor with syntax highlight
  - Domain, problem and plan side-by-side
- Visible impact of heuristics
- Visible impact of actions

- Available at web-planner.herokuapp.com

- User-defined heuristics
- Selectable color schemes
- Side-by-side state-space view for comparison
- Better parsing messages
- Verify PDDL common mistakes
  - Missing/extra requirements
  - Missing free variables
  - Effect $\subseteq$ Precondition
  - …
- Larger user survey
- More planning instances available
- Define an API