



HAL
open science

Feedback on the use of PDDL solvers in an industrial R&D context

Johann Dreo

► **To cite this version:**

Johann Dreo. Feedback on the use of PDDL solvers in an industrial R&D context. Doctoral. Innovation, Design, and Engineering, Västerås [Sweden], France. 2022. hal-04110686

HAL Id: hal-04110686

<https://hal.science/hal-04110686>

Submitted on 30 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Feedback on the use of PDDL solvers in an industrial R&D context

Johann Dreo

Message



Happy planning solvers users



Not happy with:

**No on-demand binding between model
and action costs**

Debugging of domains too hard

Team

▶ 2007 - 2013 - 2020

- ▶ 1) develop solvers
- ▶ 2) applications

▶ Research & Technology

- ▶ Decision & Optimization lab.,
 - ▶ Johann Dreo,
 - ▶ Pierre Savéant.

▶ Others

- ▶ INRIA : March Schoenauer
- ▶ ONERA : Vincent Vidal

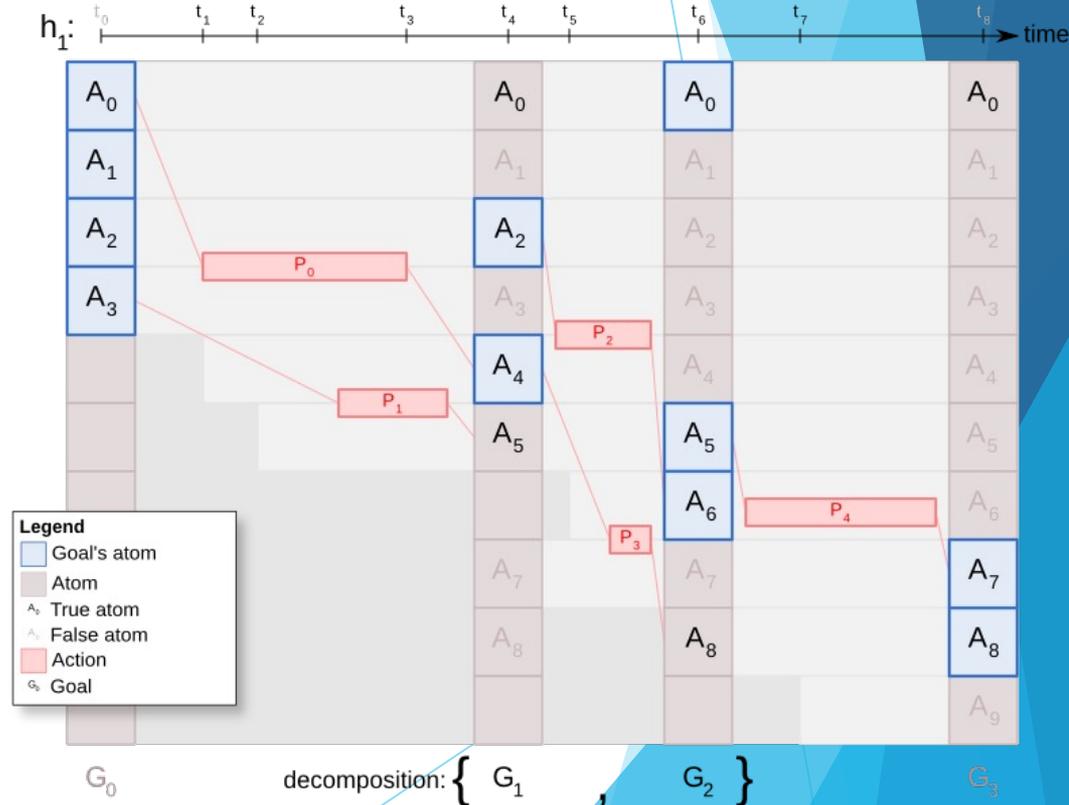
▶ Former students/postdocs

- ▶ Jacques Bibai
- ▶ Caner Candan
- ▶ Mátyás Brendel
- ▶ Mostepha R. Khouadjia
- ▶ Alexandre Quemy

Divide-and-Evolve (DAE)

▶ Divide And Evolve

- ▶ Black-box heuristic for problem decomposition
- ▶ YAHSP for subproblem solving
- ▶ 2010 : silver medal at GECCO Hummies awards
 - ▶ Competing against human planner
- ▶ 2011 : (one of the) winner in temporal satisficing track at IPC



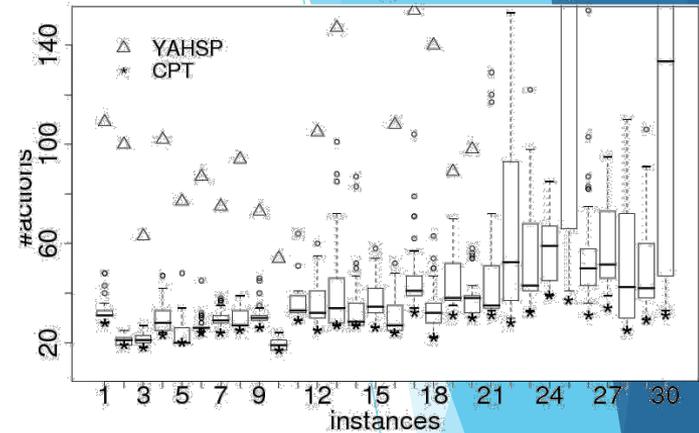
Research: 2007 - 2013

- ▶ DAE-CPT
- ▶ DAE-YAHSP2
- ▶ DAE-YAHSP3

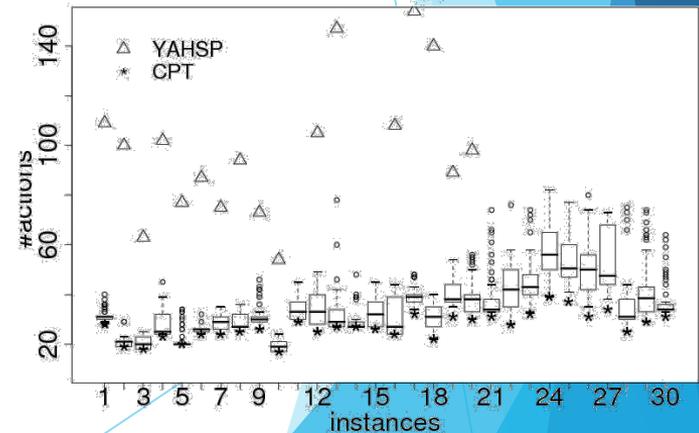
- ▶ Automatic configuration of DAE
 - ▶ Using racing
 - ▶ Using generic optimization

- ▶ Multi-objective optimization
 - ▶ And PDDL (Zeno) benchmark

Gold miner: global racing



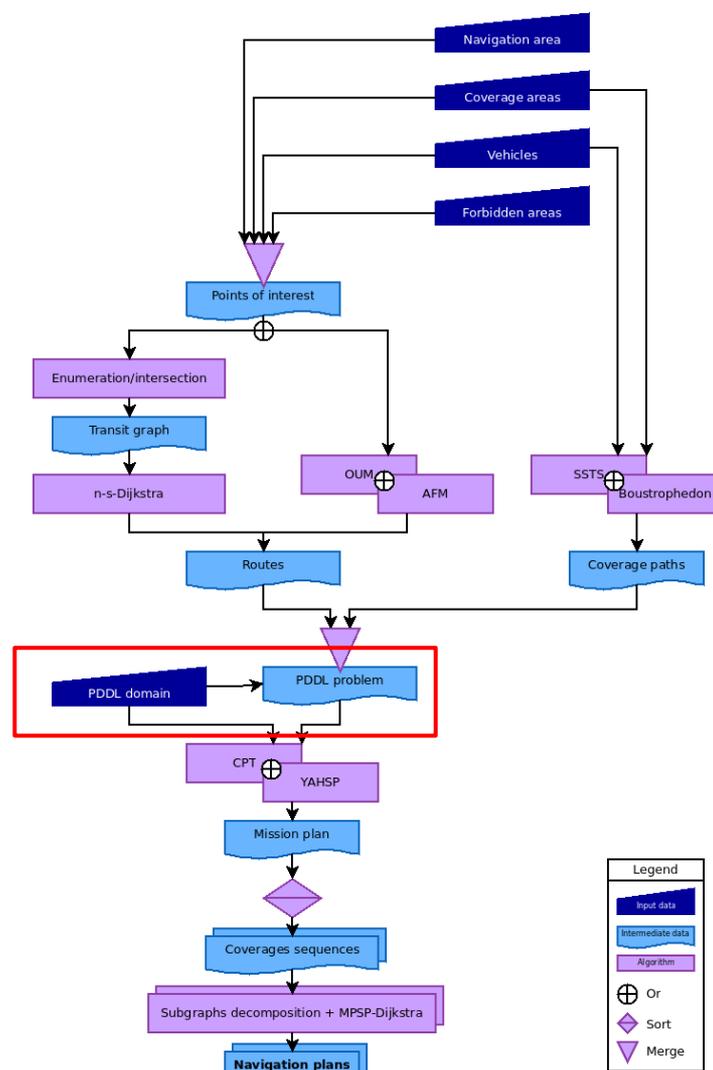
Gold miner: racing per instance



Applications: 2013 - 2020

- ▶ **Multi-vehicles mission planning**
(\approx resources displacement on geographic graphs)
 - ▶ Emergency vehicles in heavy traffic
 - ▶ Delivery during resource-scare crisis
 - ▶ Aerial drone mission planning
 - ▶ Multi-aerial drones surveillance system
 - ▶ Underwater uncrewed multi-vehicles mission planning
 - ▶ Multi-types uncrewed multi-vehicles mission planning
- ▶ **Always various actions (more than just "move", e.g. scan, pick, ...)**
- ▶ **Not always required concurrency**

Multi-vehicles planning



Well known Pro & Cons of Planning Modelling

▶ Advantages

- ▶ Optimal is always better than humans
- ▶ Satisficing is often good enough

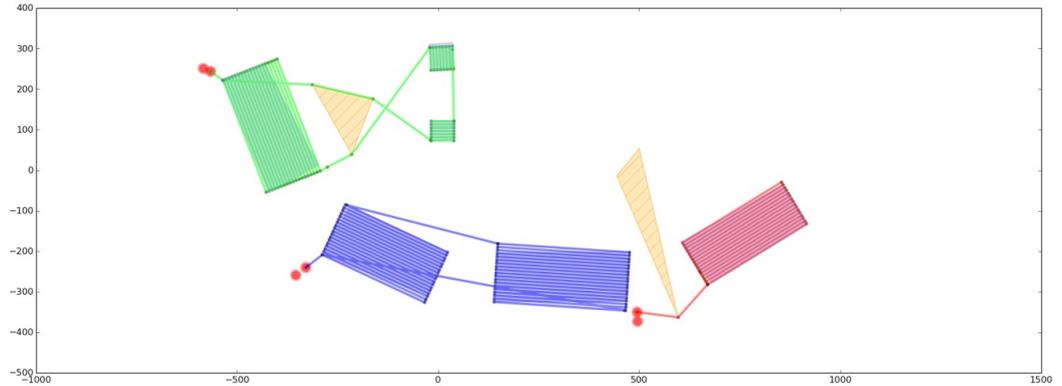
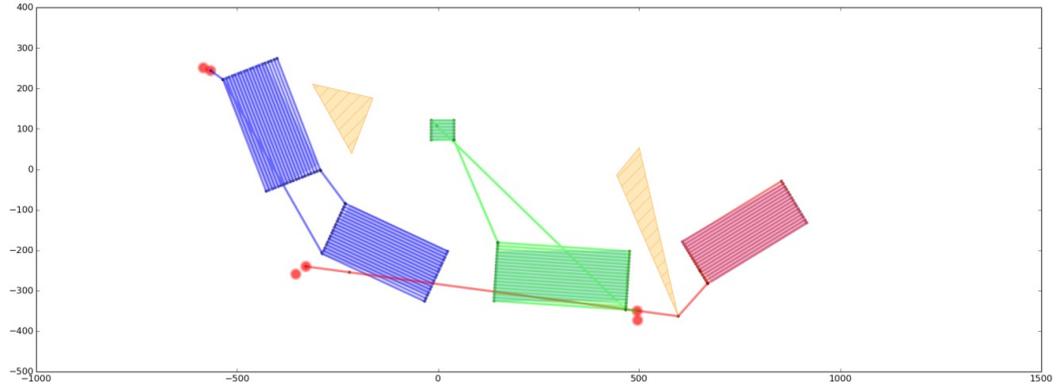
- ▶ In practice, fast enough to compute many real-life instance sizes if the model is coarse enough

▶ Drawbacks

- ▶ Poor correlation in decision space

Drawbacks of planning model

- ▶ Poor correlation in decision space



Well known Pro & Cons of Planning Modelling

▶ Advantages

- ▶ Optimal is always better than humans
- ▶ Satisficing is often good enough
- ▶ In practice, fast enough to compute many real-life instance sizes if the model is coarse enough

▶ Drawbacks

- ▶ Poor correlation in decision space
- ▶ Optimal runtime is unpredictable
- ▶ Satisficing quality is unpredictable
- ▶ Do not scale well with instance size

User expectations VS reality

▶ Expectations

- ▶ Model once, solve many instances
- ▶ Model by domain experts, of-the-shelf solvers
- ▶ Atomic costs computation is the bottleneck
- ▶ Separation of model and solver

▶ Reality

- ▶ Problem landscapes diversity is HUGE
- ▶ Domain experts is a VERY scarce resource
- ▶ Hard to embed in solvers/models for on-demand computation.
- ▶ PDDL itself is not scalable enough

Example PDDL domain

```
(define (domain domain_surveillance_v0)
  (:requirements :typing :durative-actions)
  (:types
    Vehicle - Object
    Type_A - Vehicle
    Type_B - Vehicle
    Location - Object
    Point Area - Location
    Start Wait - Point
  )
  (:functions
    ; Time to move from x to y
    (move_time ?x - Location ?y - Location)
    ; Time to scan area m
    (scan_time ?m - Area)
  )
)
```

```
(:predicates
  ; Vehicle d is located at location x
  (located ?d - Vehicle ?x - Location)
  ; Vehicle d is not already busy
  (available ?d - Vehicle)
  ; Sensor activation
  (sensor_on ?d - Vehicle)
  (sensor_off ?d - Vehicle)
  ; Vehicle position relatively to surface.
  (underwater ?d - Vehicle)
  (abovewater ?d - Vehicle)
  ; Area m is scanned / not scanned
  (revealed ?m - Area) ; scanned
  (concealed ?m - Area) ; not scanned
  ; NOTE: At init an goal, we cannot use the "not" operator,
  ; thus we must have two predicates for the state of areas
  ; in order to specify a switch of status:
  ; areas are concealed in the init and revealed in the goal.
)
)
```

Example PDDL domain

```
; dive
(:durative-action dive
  :parameters (?d - Vehicle)
  :duration (= ?duration 10)
  :condition (and
    (at start (available ?d) ) ; the vehicle is not doing something else
    (at start (abovewater ?d)) ; the vehicle is above surface
    (at start (sensor_off ?d)) ; not allowed to use sensor while diving
    ; NOTE: you cannot use the "not" operator in a condition.
  )
  :effect (and
    (at start (not (available ?d))) ; the vehicle is doing something from the very beginning
    (over all (not (available ?d))) ; the vehicle is doing something during the action

    (at end (available ?d) ) ; the vehicle is no more doing something
    (at end (underwater ?d) ) ; the vehicle is under water
    (at end (not (abovewater ?d))) ; the vehicle is not above surface
  )
) ; dive
; [...]
```

Example PDDL instance

```
(define (problem {{NAME}}) (:domain
  surveillance_v0)

  (:objects [...] )

  (:init

    {{#VEHICLES}}

    (located {{NAME}} {{TAKEOFF}})

    (available {{NAME}})

    {{/VEHICLES}}

    {{#MISSIONS}}

    (concealed {{NAME}})

    {{/MISSIONS}}

    {{#MOVES}}

    (= (move_time {{FROM}} {{TO}})
      {{TIME}})

    {{/MOVES}}

    {{#MISSIONS}}

    (= (scan_time {{NAME}})
      {{SCAN_TIME}})

    {{/MISSIONS}}

  ) ; init
```

```
(:goal

  (and

    {{#MISSIONS}}

    (revealed {{NAME}})

    {{/MISSIONS}}

  )

) ; goal

(:metric minimize (total-time))

)
```

Example PDDL plan

; Time 0.024

; Length 11

; Makespan 41.81

0 : (move_drone d1 s1 m3) [11.95]

0 : (move_drone d2 s2 m1) [10]

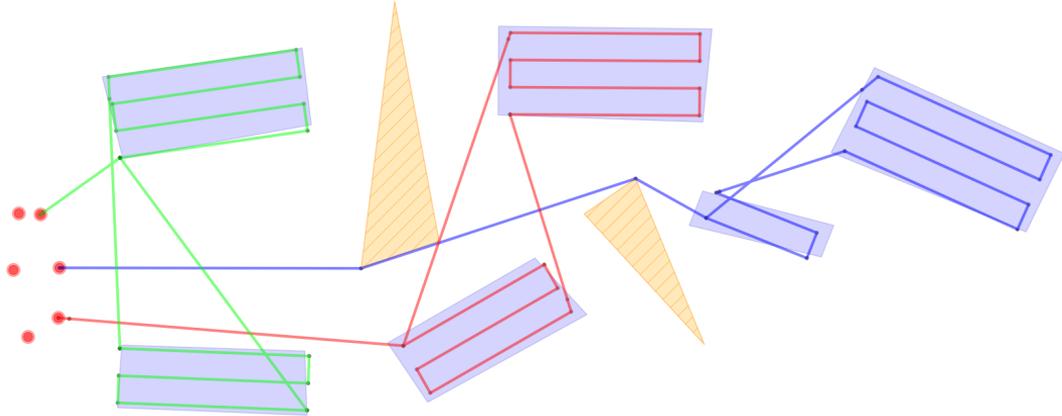
0 : (move_drone d0 s0 m2) [7.28]

7.28 : (scan_mission d0 m2) [24]

10 : (scan_mission d2 m1) [8]

11.95 : (scan_mission d1 m3) [32]

18 : (move_drone d2 m1 m0) [7.81]



Classical modeller story line

- ▶ **Start from a close-enough existing PDDL domain**
- ▶ **Rename objects and predicates**
- ▶ **Try to add new actions and constraints**
 - ▶ Parse error in solver, but error message does not help
 - ▶ Try other solvers, various error messages (or none)
 - ▶ Random guesses
- ▶ **Make an instance template**
 - ▶ Does not scale because cost function is static
 - ▶ And pre-computations leads to combinatorial explosion
- ▶ **Call me to complain**

The problem with PDDL

- ▶ **Modelling itself is hard**

- ▶ Generic modelling problems
- ▶ Fidelity/Rapidity compromise
- ▶ Scalability of search space

- ▶ **Modelling in PDDL is even harder**

- ▶ (The language syntax is not a problem)
- ▶ (Plan validation is good enough)

- ▶ **The lack of some kind of "compiler" is the main problem**
- ▶ Using solvers to gain feedback is almost ineffective

Conclusion

▶ **Happy with:**

- ▶ Algorithmics
- ▶ Performances
- ▶ Modelling features

▶ **Not happy (to the best of our knowledge) with:**

- ▶ **No lazy binding** between model and action costs computation functions
- ▶ **Debugging of domains** is too hard
 - ▶ Ad-hoc model conversion within solvers?

johann.dreo@pasteur.fr