Deterministic and Probabilistic Planning : an AI perspective

Caroline CHANEL
Nicolas DROUGARD

DCAS / ISAE-SUPAERO
caroline.chanel@isae-supaero.fr

ISAE
Institut Supérieur de l'Aéronautique et de l'Espace
SUPAERO

Outline

AI & automation

- **Artificial Intelligence** (AI) and **Automation** are often used interchangeably for the general public
  - **Automation**: is basically making a hardware or software that is capable of doing things automatically
  - **AI**: is a science and engineering often used to describe/conceive machines (or computers) that mimic cognitive functions that humans associate with the human mind, such as **learning** and **problem solving** (Russell and Norvig, 2002)

AI & automation

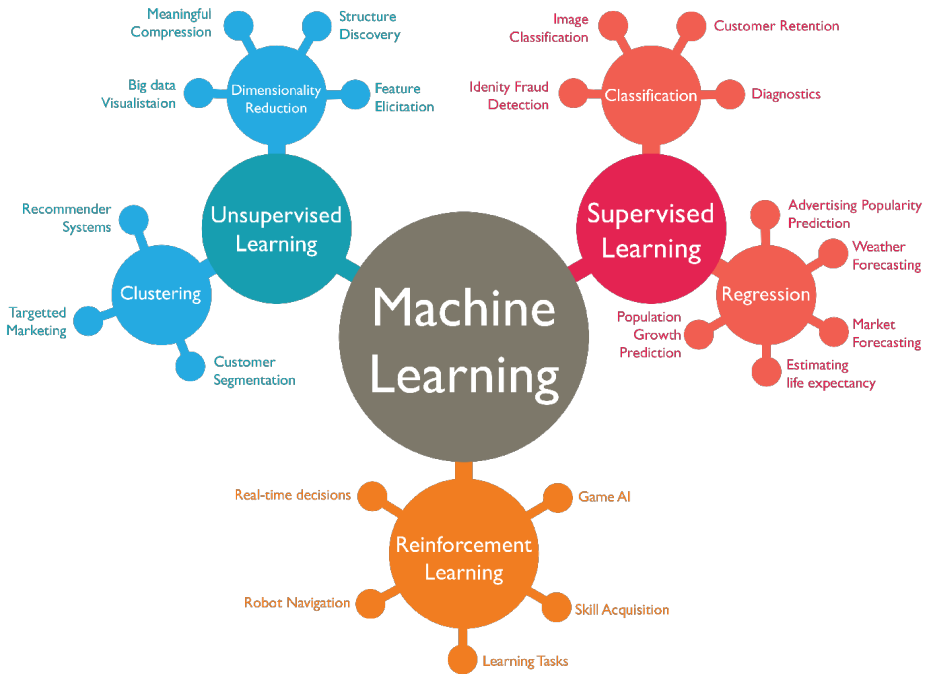
- **Artificial Intelligence** (AI) and **Automation** are often used interchangeably for the general public
  - **Automation**: is basically making a hardware or software that is capable of doing things automatically
  - **AI**: is a science and engineering often used to describe/conceive machines (or computers) that mimic cognitive functions that humans associate with the human mind, such as **learning** and **problem solving** (Russell and Norvig, 2002)
- **Automation can or can not be based on AI**
  - Industrial automation: automatic testing and control systems, equipment operation and a lot of computers solving problems

AI & automation

- **Artificial Intelligence** (AI) and **Automation** are often used interchangeably for the general public
  - **Automation**: is basically making a hardware or software that is capable of doing things automatically
  - **AI**: is a science and engineering often used to describe/conceive machines (or computers) that mimic cognitive functions that humans associate with the human mind, such as **learning** and **problem solving** (Russell and Norvig, 2002)
- **Automation can or can not be based on AI**
  - Industrial automation: automatic testing and control systems, equipment operation and a lot of computers solving problems
- Expression of **automation** are bounded by **explicit programming and rules**... as **AI** (at least for the moment) !

Learning... few examples

- Learning from examples: supervised learning, linear regression, classification ...
- Learning probabilistic models: statistical learning, learning with hidden variables ...
- Reinforcement learning: learning from rewards, policy search, ...

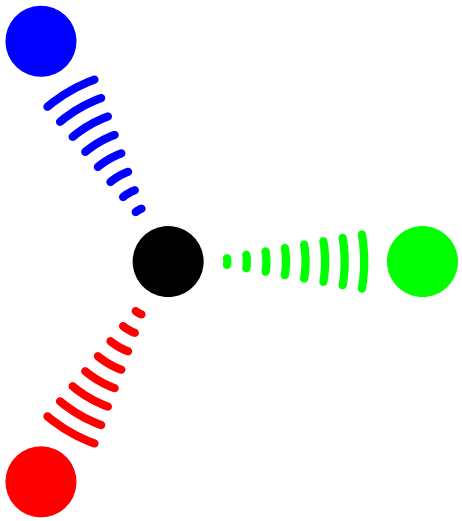


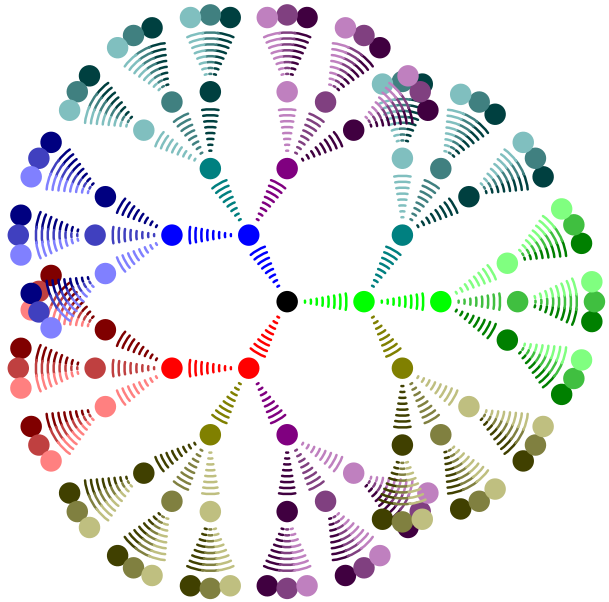
(figure credits: Abdul Rahid)

Problem solving = making decisions

(Russell and Norvig, 2002)

- Making simple decisions: short-term utility maximization, loss minimization, rules, preferences, logic ...
- Making complex decisions: long-term utility maximization, sequential decision-making problems, several reasoning steps possibly considering uncertainty on action effects at each decision step, multi-agent decision-making ...





In practice

Let's do a little exercise!

- Consider a robot with a scanner laser, a camera, an inertial measurement unit, that has to estimate its state and build a map of its environment with its location.
- It doesn't know much a priori.
- This robot uses an autopilot that should receive, for instance, a list of waypoints.
- This robot is able to plan its trajectory from a point A to a point B.
- The mission of this robot is to completely explore its environment and return to the user an occupancy map of the environment.

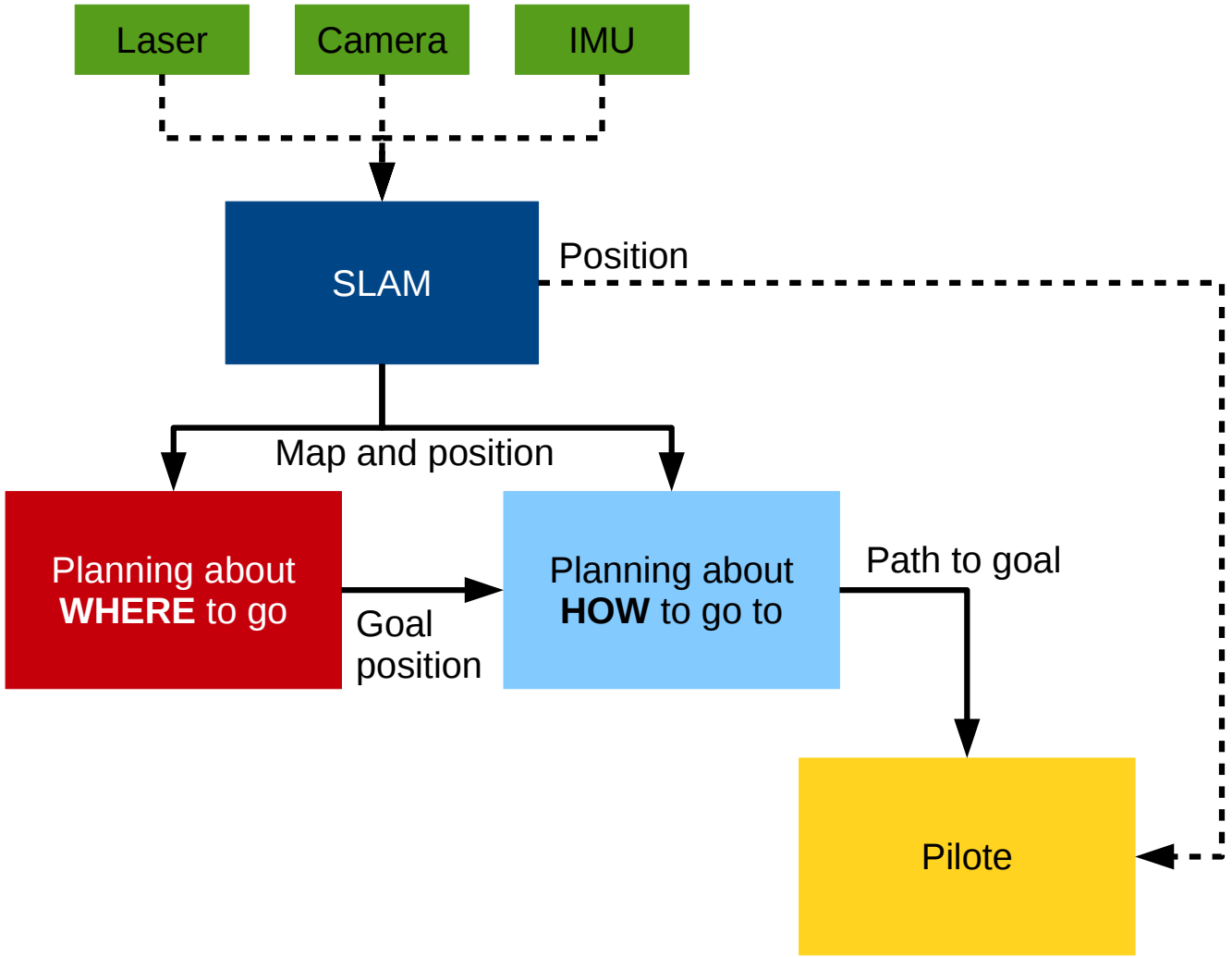
In practice

Let's do a little exercise!

- Consider a robot with a scanner laser, a camera, an inertial measurement unit, that has to estimate its state and build a map of its environment with its location.
- It doesn't know much a priori.
- This robot uses an autopilot that should receive, for instance, a list of waypoints.
- This robot is able to plan its trajectory from a point A to a point B.
- The mission of this robot is to completely explore its environment and return to the user an occupancy map of the environment.

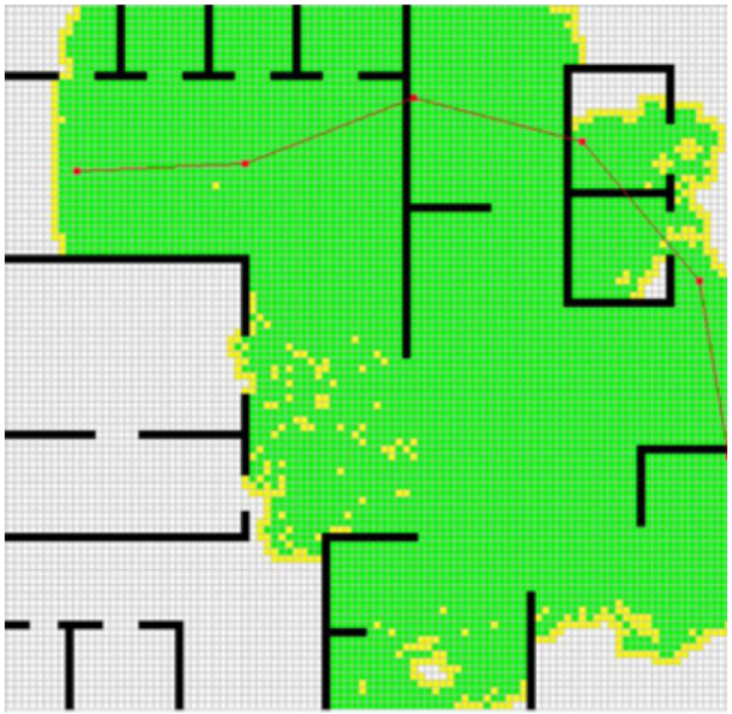
Please draw boxes (representing the functions that are essential to the completion of the mission) and arrows between them (to illustrate the inputs/outputs of the functions).

Example


Laser
Camera
IMU
SLAM
Position
Map and position
Planning about
WHERE to go
Goal
position
Planning about
HOW to go to
Path to goal
Pilote

What about the choice (decision) of where to explore?

- **How can we assess whether the environment is fully explored?**
  ⇒ **The boundary** is the segment that separates the known part of the environment from the unknown part of the environment (Keidar and Kaminka, 2012).
- **Criteria for evaluating mission performance:**
  ⇒ while at least one boundary is detected, **move towards one of these boundaries**.
  ⇒ **But which one?**
  → the closest?
  → the largest?
  → the one for which the path leading to it yields more information?
- **Conclusion: we have to decide**
  ⇒ short- or long-term planning ?



(Rooker and Birk, 2007)

But what's that word "planning" ?

From (LaValle, 2006)

"Planning" is a word that has different meanings depending on the community that uses it!

$\Rightarrow$ Control theory has traditionally been concerned with issues such as stability, feedback, and optimality, there has been a growing interest in designing algorithms that find feasible open-loop trajectories for nonlinear systems. In some of this work, the term "motion planning" has been applied, with a different interpretation from its use in robotics.

But what's that word "planning" ?

From (LaValle, 2006)

"Planning" is a word that has different meanings depending on the community that uses it!

⇒ Control theory has traditionally been concerned with issues such as stability, feedback, and optimality, there has been a growing interest in designing algorithms that find feasible open-loop trajectories for nonlinear systems. In some of this work, the term "motion planning" has been applied, with a different interpretation from its use in robotics.

⇒ In robotics the focus is on designing algorithms that generate useful motions by processing complicated geometric models. [...] The terms motion planning and trajectory planning are often used for these kinds of problems. Trajectory planning usually refers to the problem of taking the solution from a robot motion planning algorithm and determining how to move along the solution in a way that respects the mechanical limitations of the robot.

But what's that word "planning" ?

From (LaValle, 2006)

"Planning" is a word that has different meanings depending on the community that uses it!

⇒ **Control theory has traditionally been concerned with issues such as stability, feedback, and optimality, there has been a growing interest in designing algorithms that find feasible open-loop trajectories for nonlinear systems. In some of this work, the term "motion planning" has been applied, with a different interpretation from its use in robotics.**

⇒ **In robotics the focus is on designing algorithms that generate useful motions by processing complicated geometric models. [...] The terms motion planning and trajectory planning are often used for these kinds of problems. Trajectory planning usually refers to the problem of taking the solution from a robot motion planning algorithm and determining how to move along the solution in a way that respects the mechanical limitations of the robot.**

⇒ **In artificial intelligence, planning originally meant a search for a sequence of logical operators or actions that transform an initial world state into a desired goal state. Presently, planning extends beyond this to include many decision-theoretic ideas such as Markov Decision Processes, imperfect state information, and game-theoretic equilibria.**

Outline

An AI perspective : Planning

When is a plan useful and/or necessary?

- for a task that is complex to perform, or a problem that is complex to solve
- if there are any limitations due to the environment (safety or costs)
- in the context of coordination with other agents or systems
- autonomous (or embedded) systems plan their actions
- ...

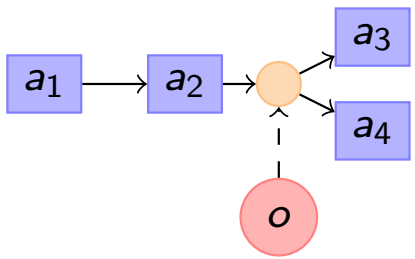
WARNING:

Depending on the knowledge of the initial state, the dynamics of the problem, the type of action, and the observability of the state, a different type of planning problem is considered.

Conditionnal plan:

Action sequence:

$a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \ldots \rightarrow a_n$



Policy:

$\pi : s \rightarrow a$

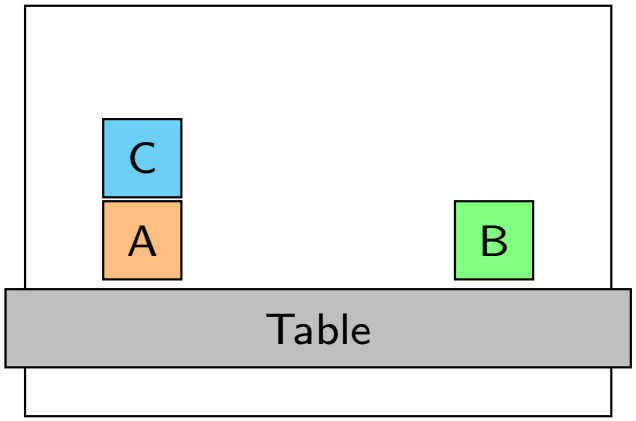
(or $\pi : b \rightarrow a$)

An AI perspective : Planning

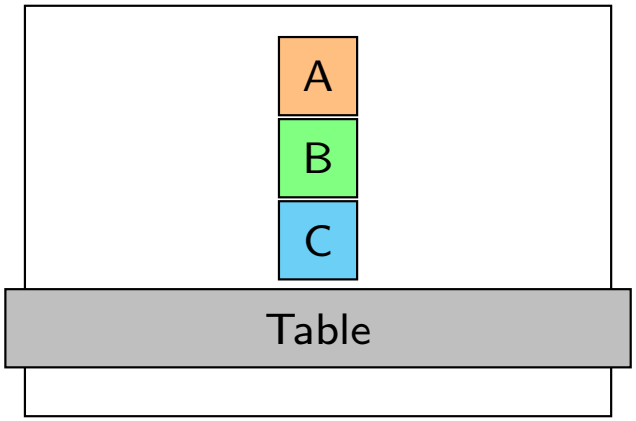
(Ghallab et al., 2004) defines planning as the process of choosing and organizing actions while anticipating their effects.

A planning problem is often represented in the following form:

- a state space (or set of tasks)
- an space of actions, whose preconditions and effects are known
- a set of final states (goal states, or completed tasks)



initial state



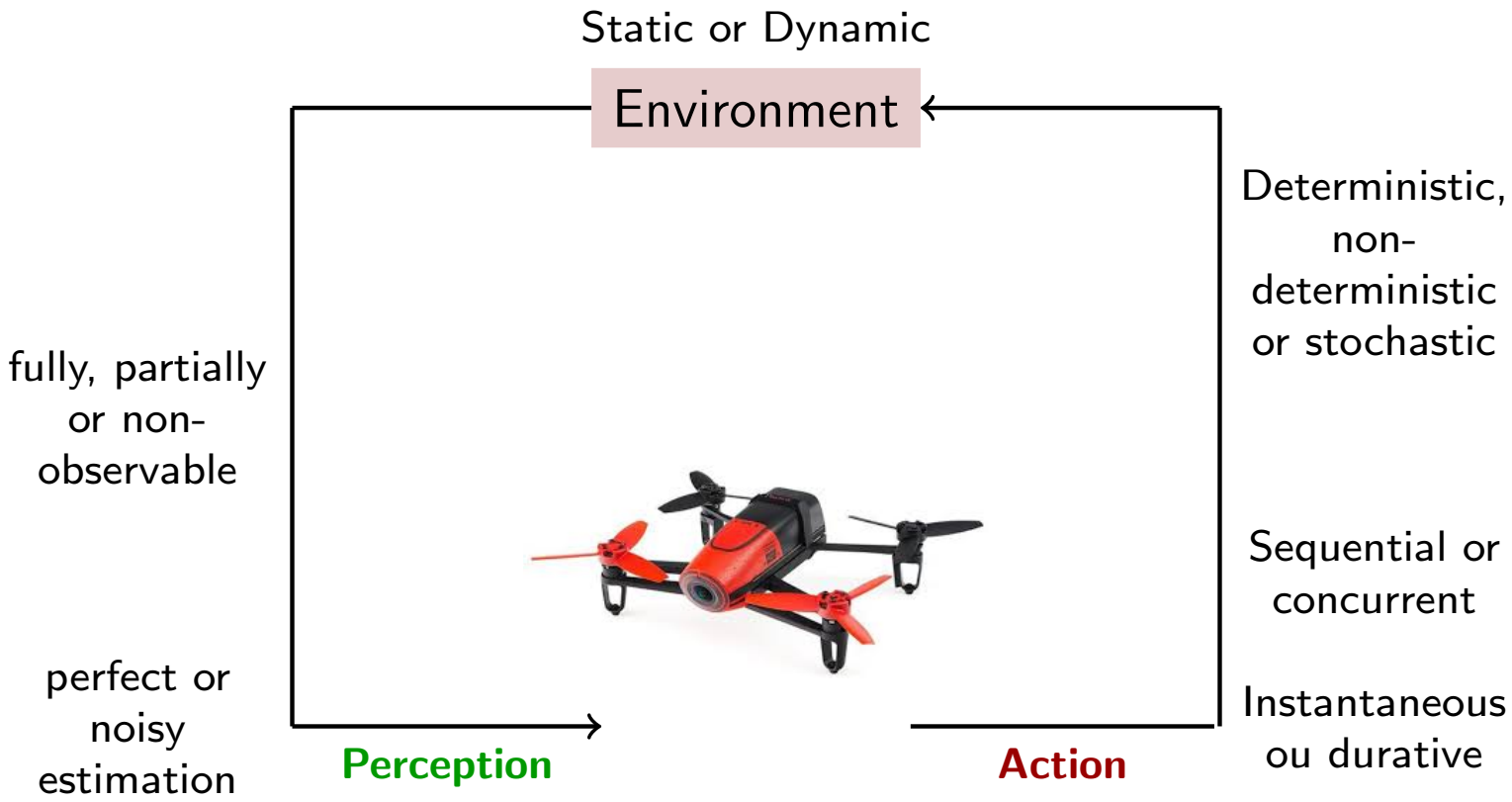
final state

⇒ **Find a plan that brings the system from the initial state to the final state.**

Some vocabulary

- **Agent:** mono or multi-agent
- **State:** complete or incomplete
  $\longrightarrow$ state describing the environment, or the knowledge of the agent
- **Actions:** that modify the environment or the knowledge on the environment and/or the state of the agent (e.g. perception)
  $\longrightarrow$ deterministic or non-deterministic (logic/stochastic)
- **Goal**: a condition satisfaction or the optimization of a criterion
  $\longrightarrow$ goal described by a final state or some knowledge,
  $\longrightarrow$ or the optimization of a cost or a utility function...
- **Reasoning:** offline or online,
  $\longrightarrow$ with full observability or partial observability
- **Plan:** sequential, partial order, conformant, contingent, conditional, policy

Action, perception – perception, action



Variations in the specification of the planning problem produce different classes of problems to be processed!

Different planning classes

dynamics: **deterministic**, non-deterministic, probabilistic
observability: **fully**, partial, zero
horizon: **finite**, infinite
objectif: **satisfiability**, optimization

- **Classical planning**
- Fully Observable Non-Deterministic (FOND) planning
- Partially Observable Non-Deterministic (POND) planning
- Conformant planning
- Markov Decision Process (MDP)
- Partially Observable MDP (POMDP)

Different planning classes

dynamics: deterministic, **non-deterministic**, probabilistic
observability: **fully**, partial, zero
horizon: **finite**, **infinite**
objectif: **satisfiability**, optimization

- Classical planning
- **Fully Observable Non-Deterministic (FOND) planning**
- Partially Observable Non-Deterministic (POND) planning
- Conformant planning
- Markov Decision Process (MDP)
- Partially Observable MDP (POMDP)

Different planning classes

dynamics: deterministic, **non-deterministic**, probabilistic
observability: fully, **partial**, zero
horizon: **finite**, **infinite**
objectif: **satisfiability**, optimization

- Classical planning
- Fully Observable Non-Deterministic (FOND) planning
- **Partially Observable Non-Deterministic (POND) planning**
  $\longrightarrow$ or **contingent planning**
- Conformant planning
- Markov Decision Process (MDP)
- Partially Observable MDP (POMDP)

Different planning classes

dynamics: deterministic, **non-deterministic**, probabilistic
observability: fully, partial, **zero**
horizon: **finite**, **infinite**
objectif: **satisfiability**, optimization

- Classical planning
- Fully Observable Non-Deterministic (FOND) planning
- Partially Observable Non-Deterministic (POND) planning
- **Conformant planning**
- Markov Decision Process (MDP)
- Partially Observable MDP (POMDP)

Different planning classes

dynamics: deterministic, non-deterministic, **probabilistic**
observability: **fully**, partial, zero
horizon: **finite**, **infinite**
objectif: satisfiability, **optimization**

- Classical planning
- Fully Observable Non-Deterministic (FOND) planning
- Partially Observable Non-Deterministic (POND) planning
- Conformant planning
- **Markov Decision Process (MDP)**
- Partially Observable MDP (POMDP)

Different planning classes

dynamics: deterministic, non-deterministic, **probabilistic**
observability: fully, **partial**, zero
horizon: **finite**, **infinite**
objectif: satisfiability, **optimization**

- Classical planning
- Fully Observable Non-Deterministic (FOND) planning
- Partially Observable Non-Deterministic (POND) planning
- Conformant planning
- Markov Decision Process (MDP)
- **Partially Observable MDP (POMDP)**

And there are other concepts...

that are not covered in this lecture.

dynamics: deterministic, non-deterministic, probabilistic
$\longrightarrow$ **explicit/implicit time**
$\longrightarrow$ **instant actions, with a duration, concurrent**
$\longrightarrow$ **agent: mono or multi-agent**
observability: fully, partial, zero
horizon: finite, infinite **Goal**: statisfiability, optimization
Goal: final state, test a knowledge,
$\longrightarrow$ **Hierarchical Task Network (HTN), script/programs**
$\longrightarrow$ **scheduling, task assignment**
contraints: precedence constraints, task parallelism
$\longrightarrow$ **application constraints, task priority**
$\longrightarrow$ **finite time horizon or limited resources**

And there are other concepts...

that are not covered in this lecture.

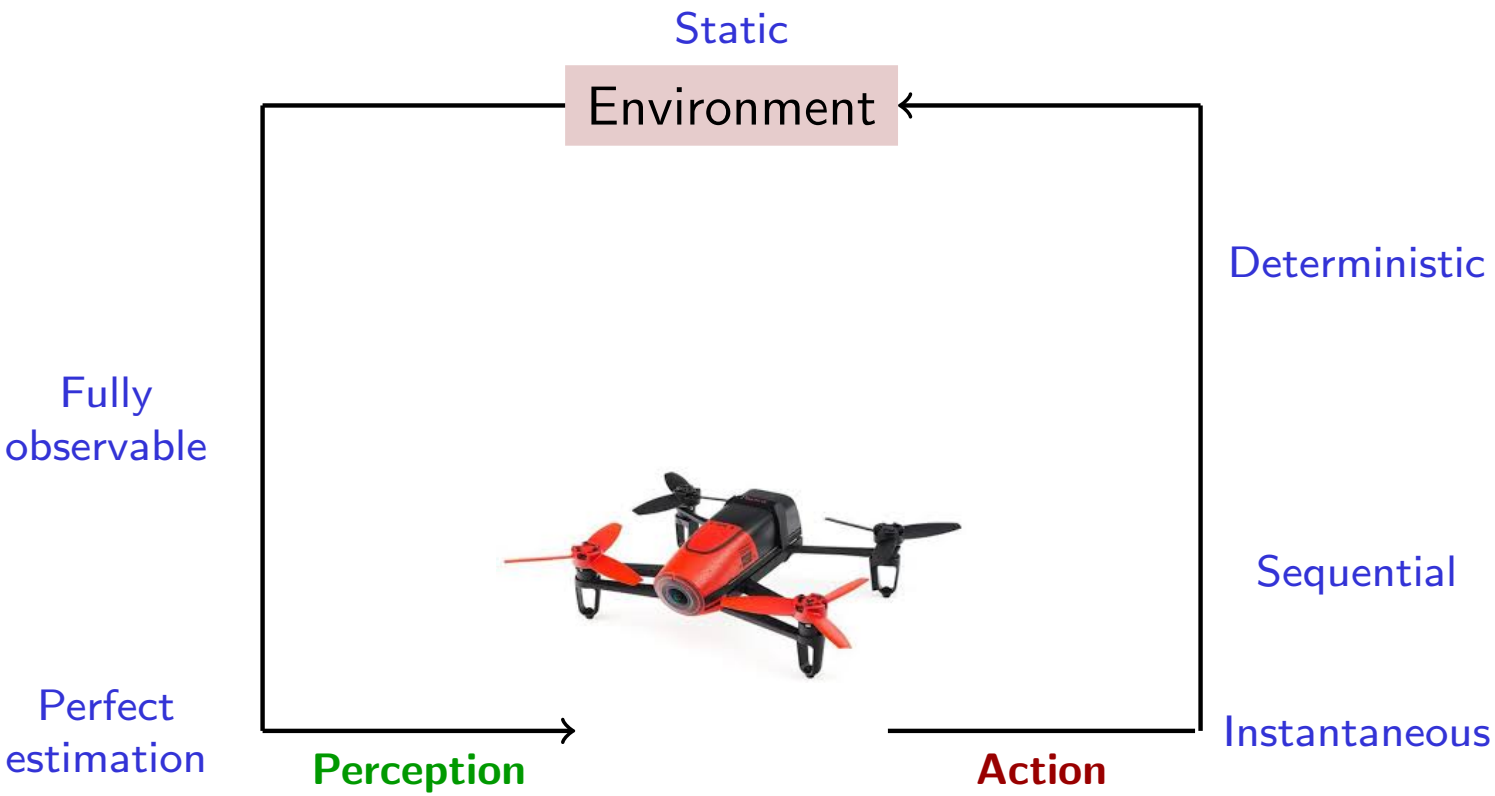
dynamics: deterministic, non-deterministic, probabilistic
$\longrightarrow$ **explicit/implicit time**
$\longrightarrow$ **instant actions, with a duration, concurrent**
$\longrightarrow$ **agent: mono or multi-agent**
observability: fully, partial, zero
horizon: finite, infinite **Goal**: statisfiability, optimization
Goal: final state, test a knowledge,
$\longrightarrow$ **Hierarchical Task Network (HTN), script/programs**
$\longrightarrow$ **scheduling, task assignment**
contraints: precedence constraints, task parallelism
$\longrightarrow$ **application constraints, task priority**
$\longrightarrow$ **finite time horizon or limited resources**

- **Scheduling**, **Constraint Satisfaction Problem (CSP)**
- **HTN planning**
- **Contraints propagation planning**
- **Game theory**

Outline

Deterministic Planning

Assumptions:



"Classical" Planning

System based on the state transition: $\Sigma = (S, A, T, G, S_0)$

- S: a bounded space of states
- A: a bounded set of actions
- T: a transition function, $T : S \times A \rightarrow S$
- G: set of goal states
- S_0: initial state

We are thus looking for a plan:

- **an action sequence that lead the system from the initial state to the goal state (or to a goal state)**

This problem can be modeled by a (directed) graph, in which a node represents a state and edges/arrows represent transitions (action consequences). **Finding a plan is like searching in this graph.**

- An **exhaustive search** in the graph can yield several solutions – a complex overall problem.
- if a criterion is defined, the **search in the graph can be guided** by this criterion. The solution is then a path in the graph that optimizes the criterion.

"Classical" Planning

System based on the state transition: $\Sigma = (S, A, T, G, S_0)$

- S: a bounded space of states
- A: a bounded set of actions
- T: a transition function, $T : S \times A \rightarrow S$
- G: set of goal states
- S_0: initial state

We are thus looking for a plan:

- **an action sequence that lead the system from the initial state to the goal state (or to a goal state)**

This problem can be modeled by a (directed) graph, in which a node represents a state and edges/arrows represent transitions (action consequences). **Finding a plan is like searching in this graph.**

- An **exhaustive search** in the graph can yield several solutions – a complex overall problem.
- if a criterion is defined, the **search in the graph can be guided** by this criterion. The solution is then a path in the graph that optimizes the criterion.

Practicals

Let's remember some graph search methods before → see *graph-search.zip* in LMS
BFS-DFS in DCG notebook

Two types of search:

- **Forward search: start with the initial state**
  $\rightarrow$ it is very similar to Breadth-first search
- Backward search: start with the final state

Forward search

Function *Forward-search*(s_0,Σ)
- Q.insert(s_0)
- **while** $Q \neq \emptyset$ **do**
  - $s \longleftarrow$ Q.removeFirst()
  - **if** $s \in Goal$ **then**
    - Plan $\longleftarrow$ mapParents.backtrack(s)
    - **return** *Plan*
  - **forall** $a \in A$ **do**
    - $s' \longleftarrow T(s, a)$
    - mapParents(s',(s, a))
    - **if** $s' \notin Q$ **then**
      - Q.insert(s')
- **return** *FAILURE*

- It is a simple algorithm, that needs to instantiate the states
- It is a **correct and complete** algorithm: it returns a solution if it exists
- Without heuristics/preferences about which actions to apply, forward search may be impractical in realistic applications in which the state space is large
- With a **heuristic** function, the actions can be ranked and the loop follows the action ranks $a_1 < a_2 < ... < a_n$ induced by the heuristic function.

Greedy search

Forward search guided by a heuristic function

Function *Greedy-search*(s_0,Σ)
- Q.insert(s_0)
- **while** $Q \neq \emptyset$ **do**
  - $s \longleftarrow$ Q.removeFirst()
  - **if** $s \in$ *Goal* **then**
    - Plan $\longleftarrow$ mapParents.backtrack(s)
    - **return** *Plan*
  - $a \longleftarrow \arg\min_{a \in A} h(T(s,a))$
  - $s' \longleftarrow T(s,a)$
  - mapParents(s',(s, a))
  - **if** $s' \notin Q$ **then**
    - Q.insert(s')
- **return** *FAILURE*

- Search guided in depth by an **heuristic** function $h()$, with no turning back
- It is a **correct but not complete** algorithm: it's not sure that a solution can be found with its mechanism.

Best-first search

In depth search guided by a heuristic function

Function *BFS(s, Σ, visited)*
- **if** $s \in Goal$ **then**
  - **return** *Plan*
- visited.add(s)
- $\mathcal{A} \leftarrow a \in A | (\text{app}(A, s) \wedge T(s, a) \notin$ visited)
- **while** $\mathcal{A} \neq \emptyset$ **do**
  - $a \leftarrow \arg\min_{a \in \mathcal{A}} h(T(s, a))$
  - $\mathcal{A} \longleftarrow \mathcal{A} - a$
  - $s' \leftarrow T(s, a)$
  - Plan $\leftarrow$ DFS(s',Σ, visited)
  - **if** *Plan* $\neq$ *FAILURE* **then**
    - Plan $\leftarrow$ a.Plan
    - **return** *Plan*
- **return** *FAILURE*

- Recursive procedure with **backtrack** in case of failure on remaining choices
- The list **visited** avoid loops that go through the same state several times
- It is a **correct and complete** algorithm
- **Animation Algo**

A* Algorithm

Function *A*(s_0,Σ)*
- closedQ $\leftarrow \emptyset$, openQ.insert(s_0)
- $g(s_0) = 0$, $f(s_0) = g(s_0) + h(s_0)$
- **while** *openQ* $\neq \emptyset$ **do**
  - $s \longleftarrow$ **openQ.popFirst()**
  - closeQ.insert(s)
  - **foreach** s' *following* $T(s, a), \forall a$ **do**
    - **if** $s' \in$ *closeQ* **then**
      - **continue**
    - **if** $s' \notin$ *openQ* **then**
      - openQ.insert(s')
    - $temp_g = g(s) + cost(s, s')$
    - **if** $temp_g \geq g(s')$ **then**
      - **continue**
    - mapParents(s',(s, a))
    - $g(s') = temp_g$
    - $f(s') = g(s') + h(s')$
    - **if** $s' \in$ *Goal* **then**
      - **return** *mapParents.backtrack(s)*
- **return** *FAILURE*

- **takes the cost of the path into account** to look for **an optimal plan**
- The function:
  $f(s) = g(s) + h(s)$, estimates the cost of a plan that pass through s, with:
  - $g(s)$: cost of the path from s_0 to s
  - $h(s)$: estimation of the cost of the path from state s until the goal
- **If the heuristic function h is admissible and monotone, the output solution of the algorithm is the optimal solution.**

Heuristic function

Admissible

means that it **never over-estimates the actual minimal cost to reach the goal**, then the algorithm A^* is optimal if we do not use the "closed queue" (closeQ).

Heuristic function

Admissible

means that it **never over-estimates the actual minimal cost to reach the goal**, then the algorithm A^* is optimal if we do not use the "closed queue" (closeQ).

Monotone

If a "closed queue" (closeQ) is used, then h has to be also monotone to make A^* algorithm optimal. It means que for each pair of adjacent states x and y,

$$h(x) \leq cost(x, y) + h(y),$$

where $cost(x, y)$ is the cost between x and y.
If $g(x)$ is the cost of the path from the initial state, we have:

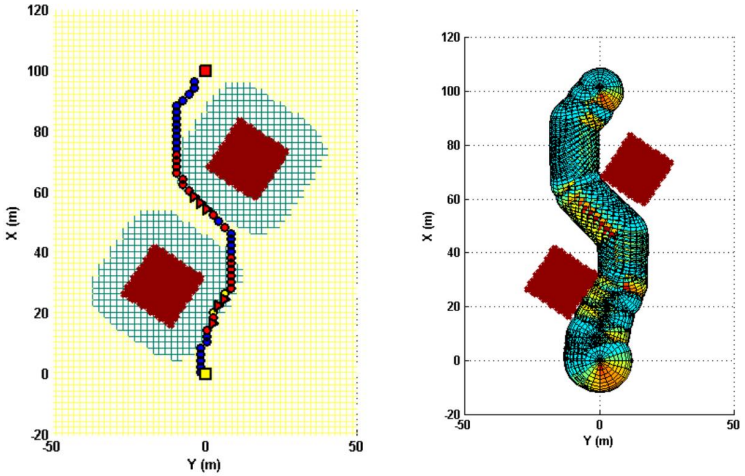
$$g(x) + h(x) \leq g(x) + cost(x, y) + h(y) = g(y) + h(y)$$

NAVPlan

Planning the guidance and navigation strategy (Watanabe et al., 2016)

Objectif: go from point A to point B as fast as possible AND as safely as possible.
Modélisation:

- State variable:
  - location X
- Actions:
  - movement direction, navigation and guidance mode;



Assumptions:

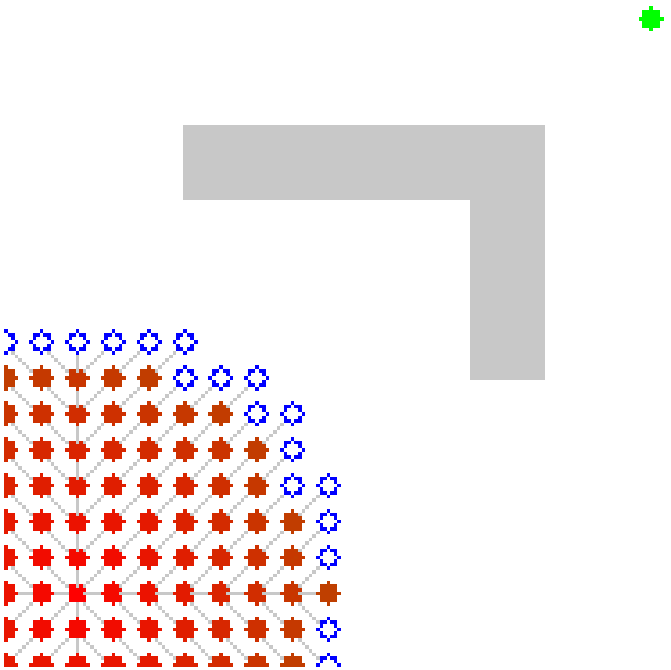
- **4 navigation modes** are available onboard:
  - **INS-only**: everytime,
  - **GPS**: only when the visual angle is greater than a threshold,
  - **Optical-flow**: only when the elevation variance is lower than a threshold,
  - **Landmark**: only when the *landmark* is in the visual field.
- **2 guidance modes** are available:
  - waypoint tracking
  - wall tracking: only when there's a wall nearby

Cost to be minimized:
- the volume of the uncertainty corridor using the A* algorithm

And if I don't know the initial state...

and if I don't have a convenient heuristic function?

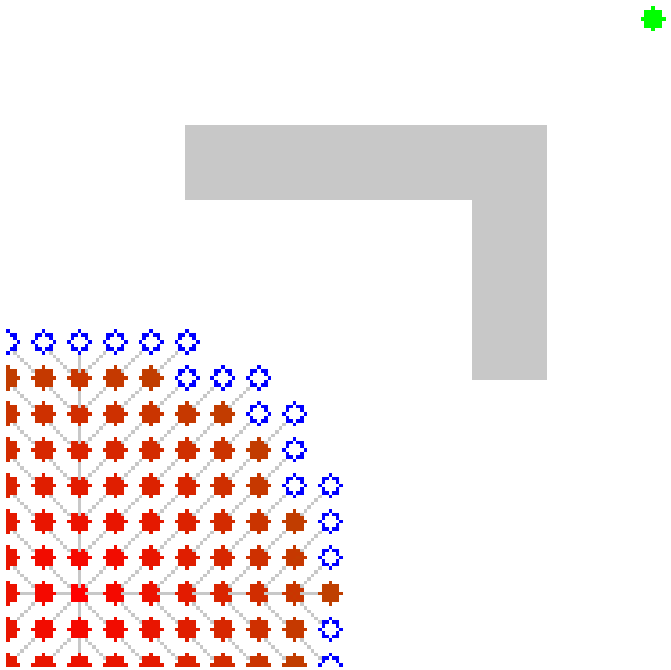
- The **Dijkstra** algorithm solves the problem of the **shortest path in a graph for any initial node**.
- a goal node is defined, and a list of the shortest paths is requested for all other nodes.
- it is a more general algorithm than the previous ones (it is more expensive).



And if I don't know the initial state...

and if I don't have a convenient heuristic function?

- The **Dijkstra** algorithm solves the problem of the **shortest path in a graph for any initial node**.
- a goal node is defined, and a list of the shortest paths is requested for all other nodes.
- it is a more general algorithm than the previous ones (it is more expensive).



Practicals

Let's check and compare those algorithms:
→ see *graph-search.zip* in LMS, A_star and Dijkstra notebooks

And the planning research is still in progress!

The newer algorithms are mostly variants of the *Depth-First search* and A*, where the **heuristic** plays a fundamental role.

- Main idea for the development of a domain-independent heuristic: **relaxation**
  $\longrightarrow$ simplify the problem data
  $\longrightarrow$ relaxation of constraints, or of effects of actions
  $\longrightarrow$ estimate the costs on the simplified problem
- we're looking for a compromise between the heuristic's computational time and its accuracy.
  $\longrightarrow$ admissible heuristic: $h(s) \leq h^*(s)$
  $\longrightarrow$ informed heuristic: provides an estimate close to the actual cost h^*
- we can **check the resulting plans** with a plan space search algorithm (not covered in this lecture).
- we use a **planning algorithm portfolio** ...
- recent algorithms: **Fast-Forward (FF)** (Hoffmann and Nebel, 2001), LAMA (Richter and Westphal, 2010), BFS(f) (Frances et al., 2018)

A general symbolic problem description

In practice, the system can be described with $n \in \mathbb{N}$ Boolean variables, often called **logical/ground atoms**, (or **predicates**, **conditions**,...): $p_1, p_2, \ldots, p_n$.

Definition: a state in practice

$\mathcal{S} \subseteq \{ T, F \}^n$: $\forall s \in \mathcal{S}$, $s = (p_1, p_2, \ldots, p_n)$ with $p_i \in \{ T, F \}$, $\forall i \leqslant n$.

A general symbolic problem description

In practice, the system can be described with $n \in \mathbb{N}$ Boolean variables, often called **logical/ground atoms**, (or **predicates**, **conditions**,...): $p_1, p_2, \ldots, p_n$.

Definition: a state in practice

$\mathcal{S} \subseteq \{ T, F \}^n$: $\forall s \in \mathcal{S}$, $s = (p_1, p_2, \ldots, p_n)$ with $p_i \in \{ T, F \}$, $\forall i \leqslant n$.

Blocks world

p_{AB} = "A on B", p_{AT} = "A on table", p_{CB} = "C on B", ...

A general symbolic problem description

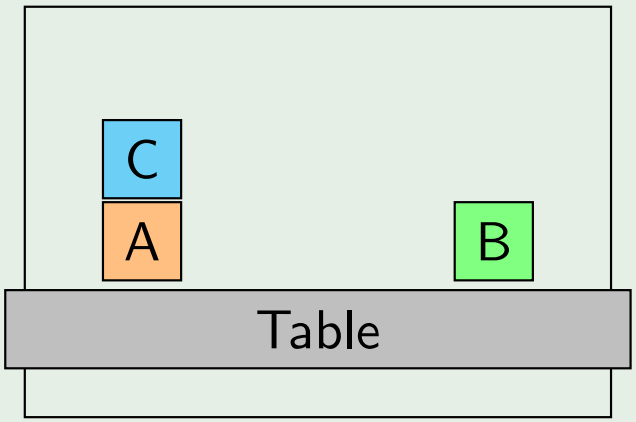
In practice, the system can be described with $n \in \mathbb{N}$ Boolean variables, often called **logical/ground atoms**, (or **predicates**, **conditions**,...): $p_1, p_2, \ldots, p_n$.

Definition: a state in practice

$\mathcal{S} \subseteq \{T, F\}^n$: $\forall s \in \mathcal{S}$, $s = (p_1, p_2, \ldots, p_n)$ with $p_i \in \{T, F\}$, $\forall i \leqslant n$.

Blocks world

p_{AB} = "A on B", p_{AT} = "A on table", p_{CB} = "C on B", ...



initial state

$p_{AB} = F$
$p_{AC} = F$
$p_{AT} = T$
$p_{BA} = F$
$p_{BC} = F$
$p_{BT} = T$
$p_{CA} = T$
$p_{CB} = F$
$p_{CT} = F$

A general symbolic problem description

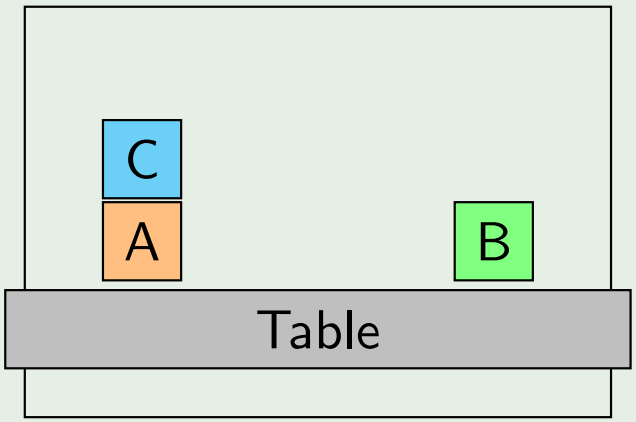
In practice, the system can be described with $n \in \mathbb{N}$ Boolean variables, often called **logical/ground atoms**, (or **predicates**, **conditions**,...): $p_1, p_2, \ldots, p_n$.

Definition: a state in practice

$\mathcal{S} \subseteq \{T, F\}^n$: $\forall s \in \mathcal{S}$, $s = (p_1, p_2, \ldots, p_n)$ with $p_i \in \{T, F\}$, $\forall i \leqslant n$.

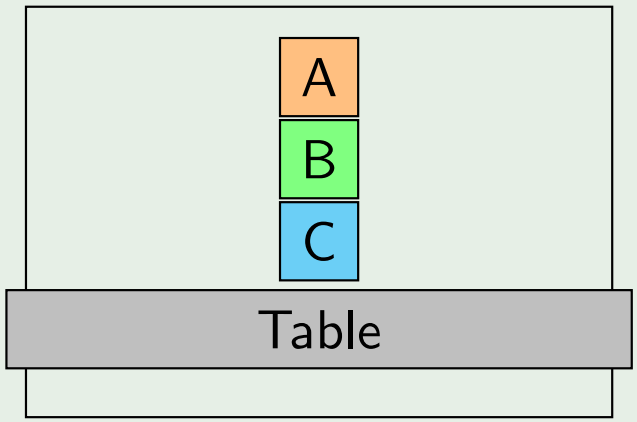
Blocks world

p_{AB} = "A on B", p_{AT} = "A on table", p_{CB} = "C on B", ...



initial state

$p_{AB} = F$
$p_{AC} = F$
$p_{AT} = T$
$p_{BA} = F$
$p_{BC} = F$
$p_{BT} = T$
$p_{CA} = T$
$p_{CB} = F$
$p_{CT} = F$



final state

$p_{AB} = T$
$p_{AC} = F$
$p_{AT} = F$
$p_{BA} = F$
$p_{BC} = T$
$p_{BT} = F$
$p_{CA} = F$
$p_{CB} = F$
$p_{CT} = T$

Problem description (STRIPS)

If applicable, an **action** (or **operator**) changes the state of the system
$\rightarrow$ some atoms become (or remain) true, some of them false.

Problem description (STRIPS)

If applicable, an **action** (or **operator**) changes the state of the system
$\rightarrow$ some atoms become (or remain) true, some of them false.

An action is entirely defined by three sets of atoms:

- **preconditions**: the atoms that must be true to apply the action,
- action's **effects**:
  - **"add" list**: the atoms that become true applying the action,
  - **"delete" list**: the atoms that become false under the action.

Problem description (STRIPS)

If applicable, an **action** (or **operator**) changes the state of the system
$\rightarrow$ some atoms become (or remain) true, some of them false.

An action is entirely defined by three sets of atoms:

- **preconditions**: the atoms that must be true to apply the action,
- action's **effects**:
  - **"add" list**: the atoms that become true applying the action,
  - **"delete" list**: the atoms that become false under the action.

Action (or operator) "put-C-on-table"

precondition:

$\{ p_{Cfree} \}$

with

$p_{Cfree} = \neg p_{AC} \wedge \neg p_{BC}$

add list: $\{ p_{CT}, p_{Afree} \}$

delete list: $\{ p_{CA} \}$

Problem description (STRIPS)

If applicable, an **action** (or **operator**) changes the state of the system
$\rightarrow$ some atoms become (or remain) true, some of them false.

An action is entirely defined by three sets of atoms:

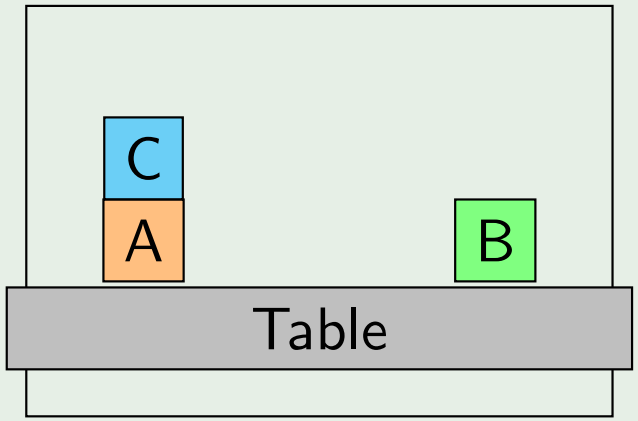
- **preconditions**: the atoms that must be true to apply the action,
- action's **effects**:
  - **"add" list**: the atoms that become true applying the action,
  - **"delete" list**: the atoms that become false under the action.

Action (or operator) "put-C-on-table"

precondition:
$\{ p_{Cfree} \}$
with
$p_{Cfree} = \neg p_{AC} \wedge \neg p_{BC}$

add list: $\{ p_{CT}, p_{Afree} \}$

delete list: $\{ p_{CA} \}$



initial state

Problem description (STRIPS)

If applicable, an **action** (or **operator**) changes the state of the system
$\rightarrow$ some atoms become (or remain) true, some of them false.

An action is entirely defined by three sets of atoms:

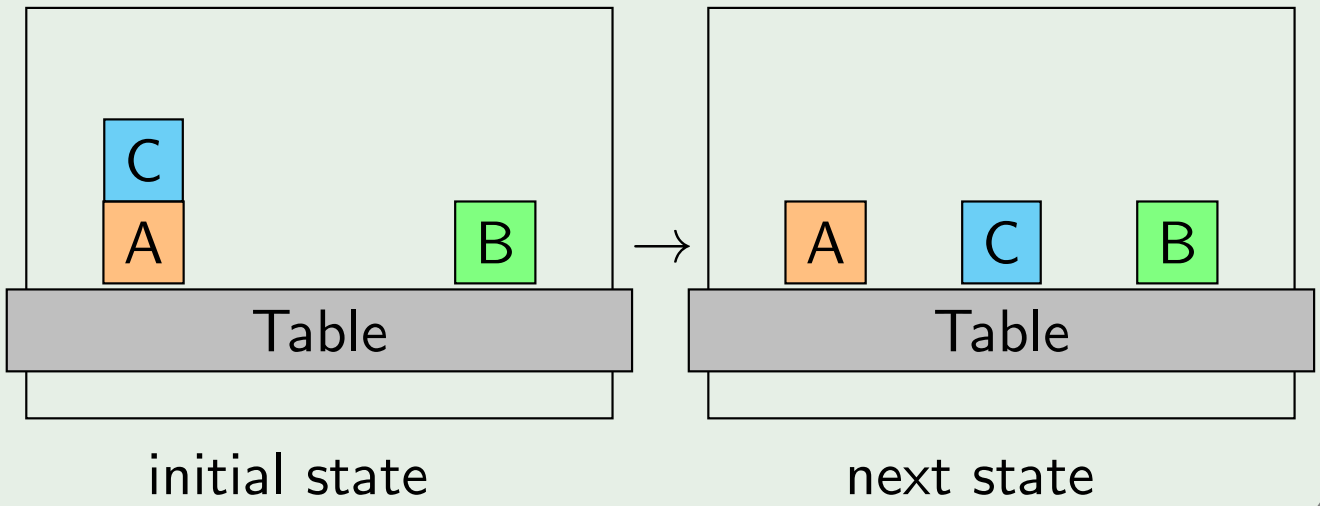
- **preconditions**: the atoms that must be true to apply the action,
- action's **effects**:
  - **"add" list**: the atoms that become true applying the action,
  - **"delete" list**: the atoms that become false under the action.

Action (or operator) "put-C-on-table"

precondition:
$\{ p_{Cfree} \}$
with
$p_{Cfree} = \neg p_{AC} \wedge \neg p_{BC}$

add list: $\{ p_{CT}, p_{Afree} \}$

delete list: $\{ p_{CA} \}$



Problem description (STRIPS)

Let us define a **STRIPS**[1] **planning task**.

Definition: STRIPS planning task

A planning task $\mathcal{P}$ is a triple $(\mathcal{A}, \mathcal{I}, \mathcal{G})$, where:

- $\mathcal{A}$ is the set of actions $a = (\mathit{precond}., \mathit{add}, \mathit{del}.)$,
- $\mathcal{I}$ is the set of all atoms that are **true in the initial state**,
- $\mathcal{G}$ is the set of atoms that are **true for each goal state**.

[1] The *Stanford Research Institute Problem Solver*, known by its acronym **STRIPS**, is an automated planner developed by R. Fikes and N. Nilsson in 1971. It also refers to the formal language of the inputs to this planner.

Problem description (STRIPS)

Let us define a **STRIPS**[1] **planning task**.

Definition: STRIPS planning task

A planning task $\mathcal{P}$ is a triple $(\mathcal{A}, \mathcal{I}, \mathcal{G})$, where:

- $\mathcal{A}$ is the set of actions $a = (\mathit{precond}., \mathit{add}, \mathit{del}.)$,
- $\mathcal{I}$ is the set of all atoms that are **true in the initial state**,
- $\mathcal{G}$ is the set of atoms that are **true for each goal state**.

Definition: (delete)-relaxed planning task

Given a planning task $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$, its **relaxation** is $\mathcal{P}' = (\mathcal{A}', \mathcal{I}, \mathcal{G})$, with $\mathcal{A}' = \{ (\mathit{precond}., \mathit{add}, \emptyset) \mid (\mathit{precond}., \mathit{add}, \mathit{del}.) \in \mathcal{A} \}$.

A delete-relaxed planning task just ignores the delete lists!

[1] The *Stanford Research Institute Problem Solver*, known by its acronym **STRIPS**, is an automated planner developed by R. Fikes and N. Nilsson in 1971. It also refers to the formal language of the inputs to this planner.

PDDL (Ghallab et al., 1998)

Planning Domain Definition Language

standard encoding **language** for **classical planning** tasks
using components:
A set-theoretic planning domain:

- **Objects**: Things in the world that interest us,
- **Predicates**: Properties of objects that we are interested in, also called propositions or atoms $(p_1, p_2, \ldots)$,
- **Actions/Operators**: Ways of changing the state of the world $(\mathcal{A})$.

A set-theoretic planning problem:

- **Initial state**: The state of the world described by a set of true propositions, that we start in $(\mathcal{I})$,
- **Goal specification**: Things (propositions/predicates) that we want to be true $(\mathcal{G})$,

PDDL (Ghallab et al., 1998)

Planning Domain Definition Language

Planning tasks specified in PDDL are separated into two files:

- A domain file for predicates and actions;
- A problem file for objects, initial state and goal specification.

A **domain** defines the planning task in a **general way**
e.g. blocks on a table
and the problem defines the problem concretely
i.e. 1 (or 100) blocks to be dispose following the goal specification.

Domain Files

Domain files look like this:

```
(define (domain <domain name here>)
   <PDDL code for predicates>
   <PDDL code for first action operator>
                  ⋮
   <PDDL code for last action operator>
)
```

<domain name> is a string that identifies the planning domain, e.g. blocks-world.

Problem Files

Problem files look like this:

```
(define (problem <problem name>)
   (:domain <domain name>)
   <PDDL code for objects>
   <PDDL code for initial state>
   <PDDL code for goal specification>
)
```

Names

- <problem name> is a string that identifies the planning task, e.g. 3-blocks problem.
- <domain name> must match the domain name in the corresponding domain file, e.g. blocksworld

Classical example: Blocks World

Towering blocks

A robot equipped with a gripper can move blocks. Initially, blocks are disposed in a given configuration.
Goal: We want the blocks to be in another configuration.

- **Objects**: The blocks,
- **Predicates**: Is block x on table? Is block x over block y? Is robot holding block x ? Is block x clear? [...]
- **Initial state**: Blocks configuration, and the gripper situation (holding some block?) [...]
- **Goal specification**: Expected blocks configuration.
- **Actions/Operators**: The gripper can pic up, put down, stack or unstack blocks

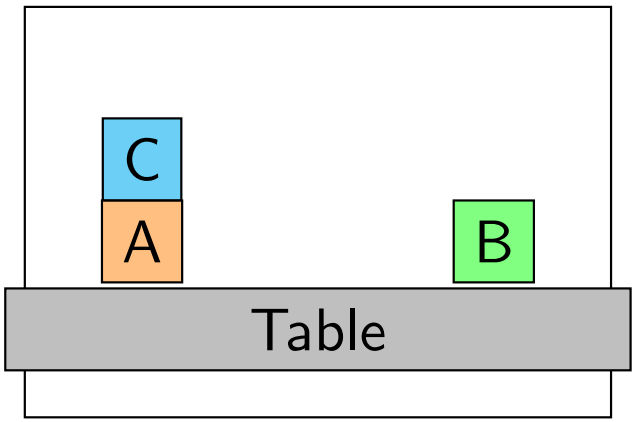
Blocks world example

Objects of the *blocks world*

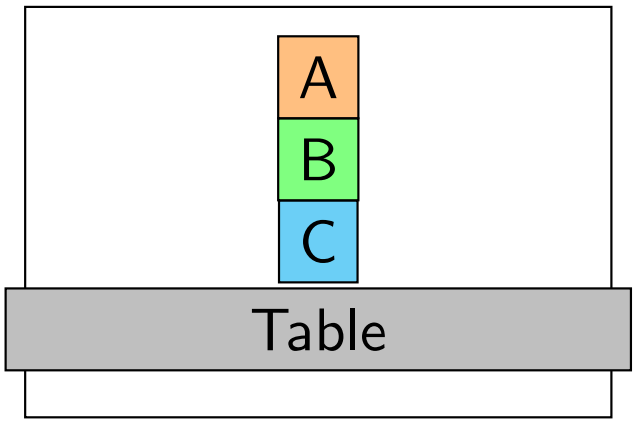
- **blocks**: `block A, block B, ..., block N`

In PDDL

```
(:objects A B C)
```



initial state



final state

Blocks world example

Predicates of the *Blocks world*

- `ON-TABLE(x)`: true iff block x is on the table
- `CLEAR(x)`: true iff block x does not have any other block over it
- `HOLDING(x)`: true iff gripper holds block x
- `ON(x,y)`: true iff block x is on block y

In PDDL

```
(:predicates (clear ?x) (on-table ?x)
   (holding ?x) (on ?x ?y)
)
```

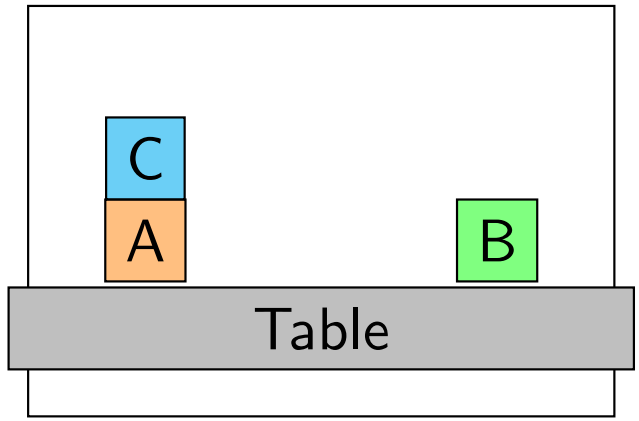
Blocks world example

Initial state of the *Blocks world*

- `ON-TABLE(A)` and `ON-TABLE(B)` are true.
- `CLEAR(B)`, `CLEAR(C)` are true.
- `ON(C,A)` is true.

In PDDL

```
(:init (on-table A)
   (on-table B)
   (on C A)
   (clear C)
   (clear B))
```



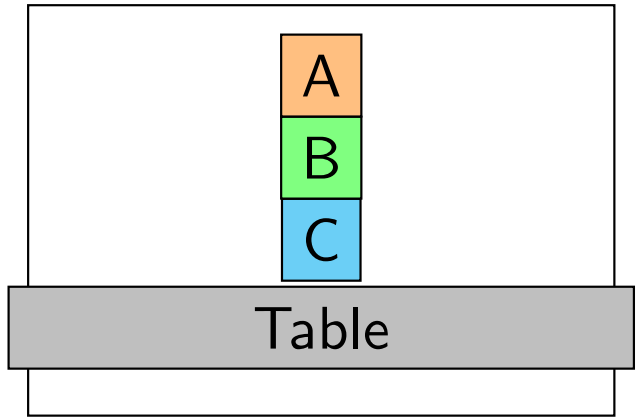
Blocks world example

Goal state specification of the *Blocks world*

- `ON-TABLE(C)` is true.
- `CLEAR(A)` is true.
- `ON(B,C)` and `ON(A,B)` are true.

In PDDL

```
(:goal (on-table C)
   (on A B)
   (on B C)
   (clear A))
```



Blocks world example

Movement action/operator of the *Blocks World*

- **Description**: The gripper can unstack block X from block Y
- **Parameters**: block X, block Y
- **Precondition**: `CLEAR(X)` and `ON(X,Y)` may be true.
- **Effect**: `HOLDING(X)` and `CLEAR(Y)`becomes true. `CLEAR(X)` and `ON(X,Y)`becomes false. Everything else doesn't change.

In PDDL

```
(:action unstack
   :parameters (?x ?y)
   :precondition (and (clear ?x) (on ?x ?y)
   :effect (and (clear ?y) (holding ?x)
                (not (clear ?x)) (not(on ?x ?y)))
```

Blocks world example

Movement action/operator of the *Blocks World*

- **Description**: The gripper can stack block X over block Y
- **Parameters**: block X, block Y
- **Precondition**: `CLEAR(Y)` and `HOLDING(X)` may be true.
- **Effect**: `ON(X,Y)` and `CLEAR(X)` becomes true. `CLEAR(Y)` and `HOLDING(X)` becomes false. Everything else doesn't change.

In PDDL

```
(:action stack
   :parameters (?x ?y)
   :precondition (and (clear ?y) (holding ?x)
   :effect (and (not (clear ?y) (not (holding ?x))
                (clear ?x) (on ?x ?y))
```

Blocks world example

Movement action/operator of the *Blocks World*

- **Description**: The gripper pick up from table block X
- **Parameters**: block X
- **Precondition**: `CLEAR(X)` and `ON-TABLE(X)` may be true.
- **Effect**: `HOLDING(X)` becomes true. `CLEAR(X)` and `ON-TABLE(X)` becomes false. Everything else doesn't change.

In PDDL

```
(:action pickup
   :parameters (?x)
   :precondition (and (clear ?x) (on-table ?x)
   :effect (and (not (clear ?x)) (not (on-table ?x))
                (holding ?x))
```

Blocks world example

Movement action/operator of the *Blocks World*

- **Description**: The gripper put block X on table
- **Parameters**: block X
- **Precondition**: `HOLDING(X)` may be true.
- **Effect**: `ON-TABLE(X)` and `CLEAR(X)` becomes true. `HOLDING(X)` becomes false. Everything else doesn't change.

In PDDL

```
(:action putdown
   :parameters (?x)
   :precondition (and (holding ?x))
   :effect (and (clear ?x) (on-table ?x)
                (not (holding ?x)))
```

Fast Forward (FF) solver

Fast Forward solver
`https://fai.cs.uni-saarland.de/hoffmann/ff.html`

Fast Forward is a solver for **classical planning** tasks developed by (Hoffmann and Nebel, 2001). It uses:

- it takes a planing problem in **PDDL description**
- GRAPHPLAN (Blum and Furst, 1995) computing an **heuristic** using the *(delete)-relaxed task*;
- Enforced Hill-climbing ($\sim$ Best-First Search) based on this heuristic.

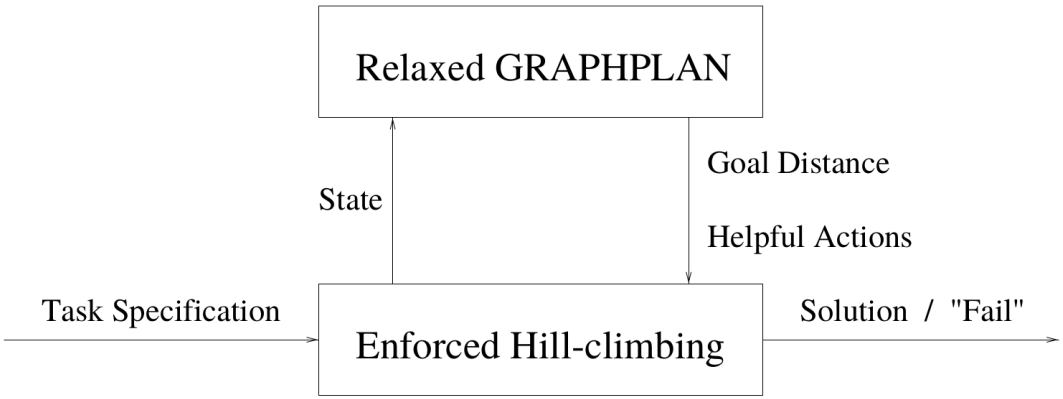
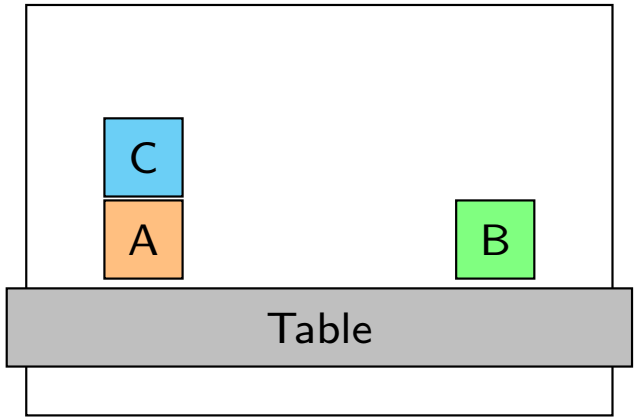


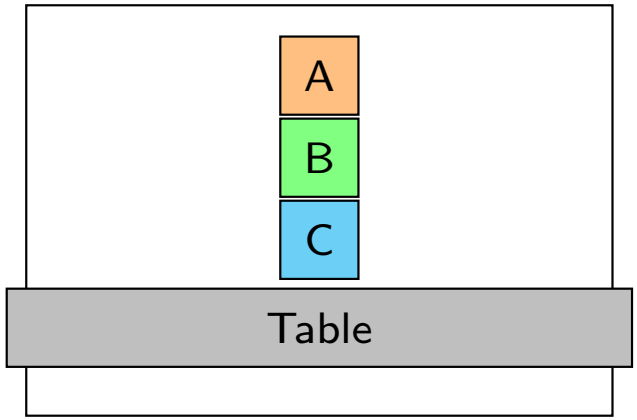
Figure 1: FF's base system architecture.

FF demo for Blocks world example

```
c.ponzoni@port-chanel-l: ~/Work/cours_planif/cours_planif_DNIA302/demo
File  Edit  View  Search  Terminal  Help

c.ponzoni@port-chanel-l:~/Work/cours_planif/cours_planif_DNIA302/demo$ ./ff -o blocksworld-domain.pddl
-f blocksworld-prob-3-blocks.pddl

ff: parsing domain file
domain 'BLOCKSWORLD' defined
 ... done.
ff: parsing problem file
problem 'BLOCKSWORLD-3-BLOCKS' defined
 ... done.



Cueing down from goal distance:    2 into depth [1]
                                   1            [1]
                                   0

Cueing down from goal distance:    2 into depth [1]
                                   1            [1]
                                   0

Cueing down from goal distance:    2 into depth [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: UNSTACK C A
        1: PUTDOWN C
        2: PICKUP B
        3: STACK B C
        4: PICKUP A
        5: STACK A B


time spent:    0.00 seconds instantiating 24 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 18 facts and 24 actions
               0.00 seconds creating final representation with 18 relevant facts
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 10 states, to a max depth of 1
               0.00 seconds total time

c.ponzoni@port-chanel-l:~/Work/cours_planif/cours_planif_DNIA302/demo$
```



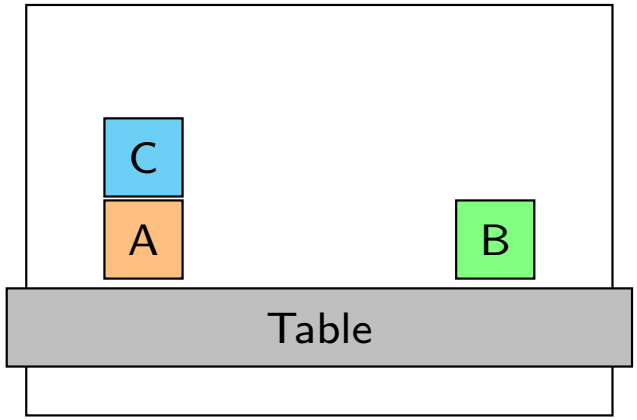
Initial state



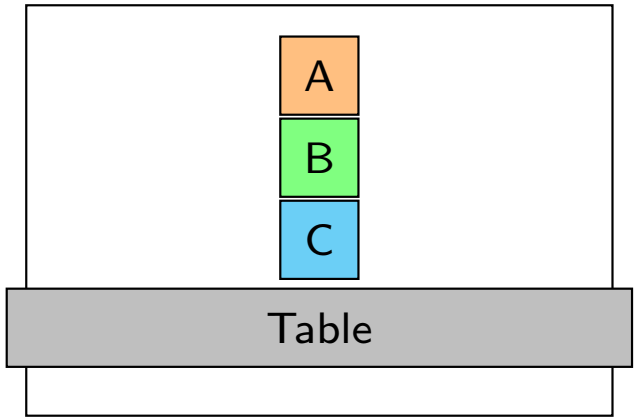
Final state

FF demo for Blocks world example

```
c.ponzoni@port-chanel-l: ~/Work/cours_planif/cours_planif_DNIA302/demo
File  Edit  View  Search  Terminal  Help

c.ponzoni@port-chanel-l:~/Work/cours_planif/cours_planif_DNIA302/demo$ ./ff -o blocksworld-domain.pddl
-f blocksworld-prob-3-blocks.pddl

ff: parsing domain file
domain 'BLOCKSWORLD' defined
 ... done.
ff: parsing problem file
problem 'BLOCKSWORLD-3-BLOCKS' defined
 ... done.



Cueing down from goal distance:    2 into depth [1]
                                   1            [1]
                                   0

Cueing down from goal distance:    2 into depth [1]
                                   1            [1]
                                   0

Cueing down from goal distance:    2 into depth [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: UNSTACK C A
        1: PUTDOWN C
        2: PICKUP B
        3: STACK B C
        4: PICKUP A
        5: STACK A B


time spent:    0.00 seconds instantiating 24 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 18 facts and 24 actions
               0.00 seconds creating final representation with 18 relevant facts
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 10 states, to a max depth of 1
               0.00 seconds total time

c.ponzoni@port-chanel-l:~/Work/cours_planif/cours_planif_DNIA302/demo$
```



Initial state



Final state

Many others domains can be found at:
http://www.plg.inf.uc3m.es/ipc2011-learning/attachments/Domains/

PDDL extensions

There are many `versions`[a] of PDDL and some are more `expressive` than others, i.e. they make it easier to write the planning task by allowing other ways of writing things (e.g. allowing non-Boolean values).

[a] `http://en.wikipedia.org/wiki/Planning_Domain_Definition_Language`

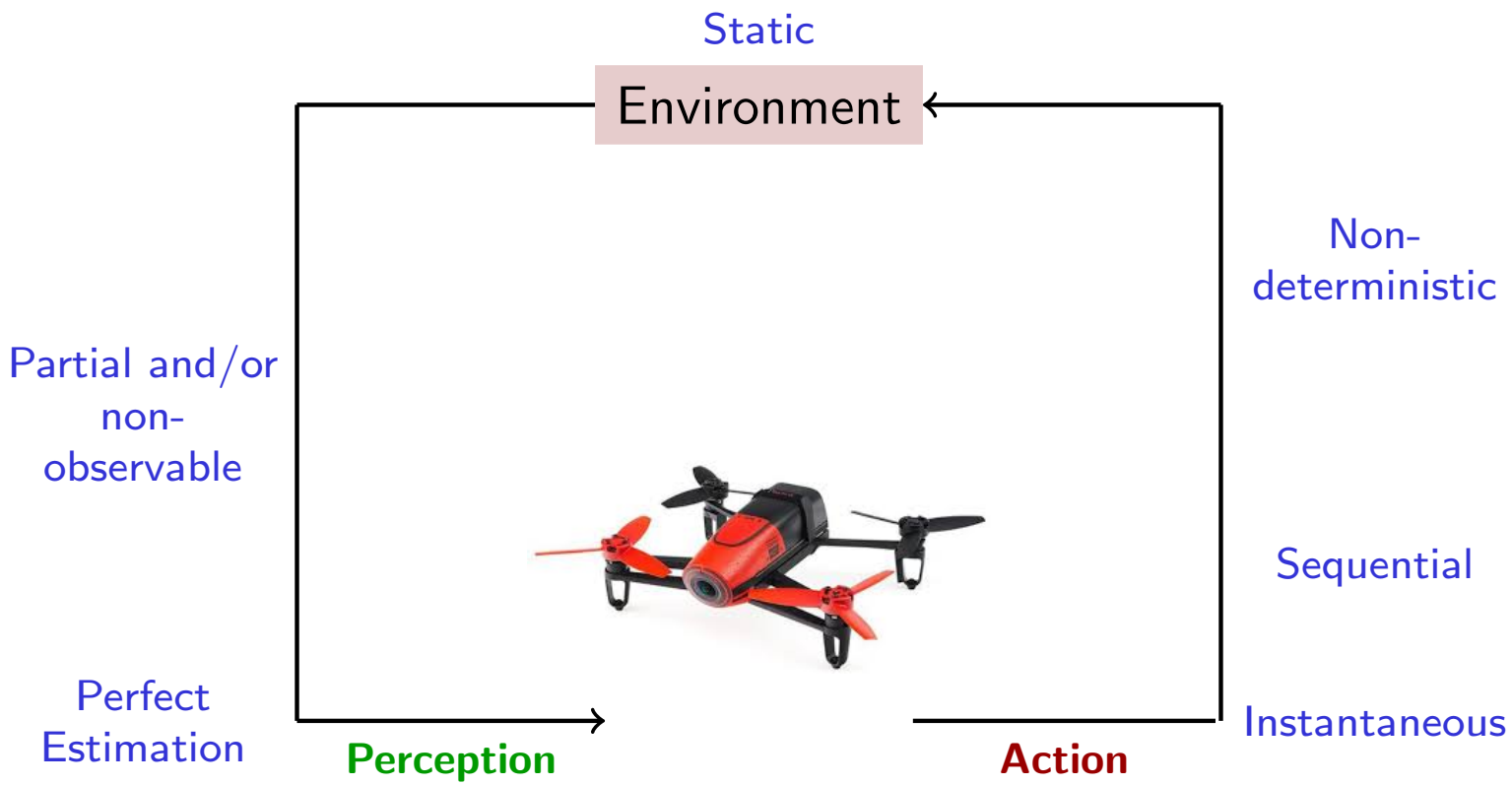
Examples

- Function **equality** can be used: `(= ?x ?y)` is true iff `?x` is `?y`
- Effects can be **universally quantified**:
  `(forall (?v1 ... ?vn) <effect>)`
- They can be **conditional**: `(when <condition> <effect>)`
- typed variables, numeric values, constraints, preferences, ...
- **probabilistic effects**, **metrics**, **actions duration**, ...

However, the task must be written in a version that the parser of the algorithm used is able to read!

Outline

Assumptions:



$\rightarrow$ **uncertainty on the initial state**

Conformant Planning

A conformant planning problem is defined with the n-uplet $\mathcal{M}$:
$\langle S, b_0, S_G, A, \varphi \rangle$,
where:

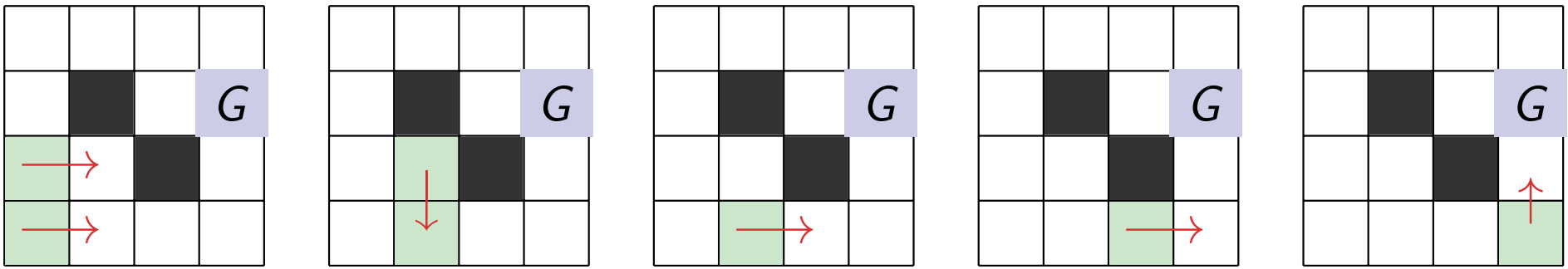
- S a set of finite states,
- $b_0 \subseteq S$ the initial belief state (non-empty set of initial states)
- $S_G \subseteq S$ the set of goal states
- A the set of actions,
- $\varphi : S \times A \rightarrow S$ is a *transition* function
  A (or several) successor state(s) $s' = \varphi(s, a)$ is (are) the result of using action a in a given state s

$b' = \varphi(b, a)$ defines the belief state b' resulting from using action a in the belief state b.

Conformant planning

- one of the simplest statements for planning with uncertainty
- a conformant planning problem is like a classical problem but with many possible initial states
- a plan is said conformant if it is a valid plan for each possible initial state.

Example with deterministic transitions: Maze Problem



Contingent planning

A contingent planning problem is defined as the n-uplet $\mathcal{M} = \langle S, S_0, S_G, A, O, \varphi, \omega \rangle$, where:

- S is a finite set of states,
- $b_0 \subseteq S$ is the initial belief state (non-empty set of initial states),
- $S_G \subseteq S$ is the set of goal states,
- A is the set of actions,
- **O is the set of sensing actions, such that $O \subseteq S$,**
- $\varphi : S \times A \to S$ is a state *transition* function.
  A successor state (or several successor states) $s' = \varphi(s, a)$ is (are) the result of the use of action a in a given state s.
- **$\omega : S \times O \to S$ is an observation (partial) function that returns for each state s a observation o if and only if it is observable, such that $(s, o) \mapsto \omega(s, o) = \{\emptyset, s\}$.**

$b' = \varphi(b, a)$ defines the belief state b' result of the use of action a in the belief state b.

$\to$ using a sensing action o to a belief state b results in a set of states that are compatible with the observation o and can be denoted by $b^o = b \cap \{\omega(s, o) \mid s \in b\}$.

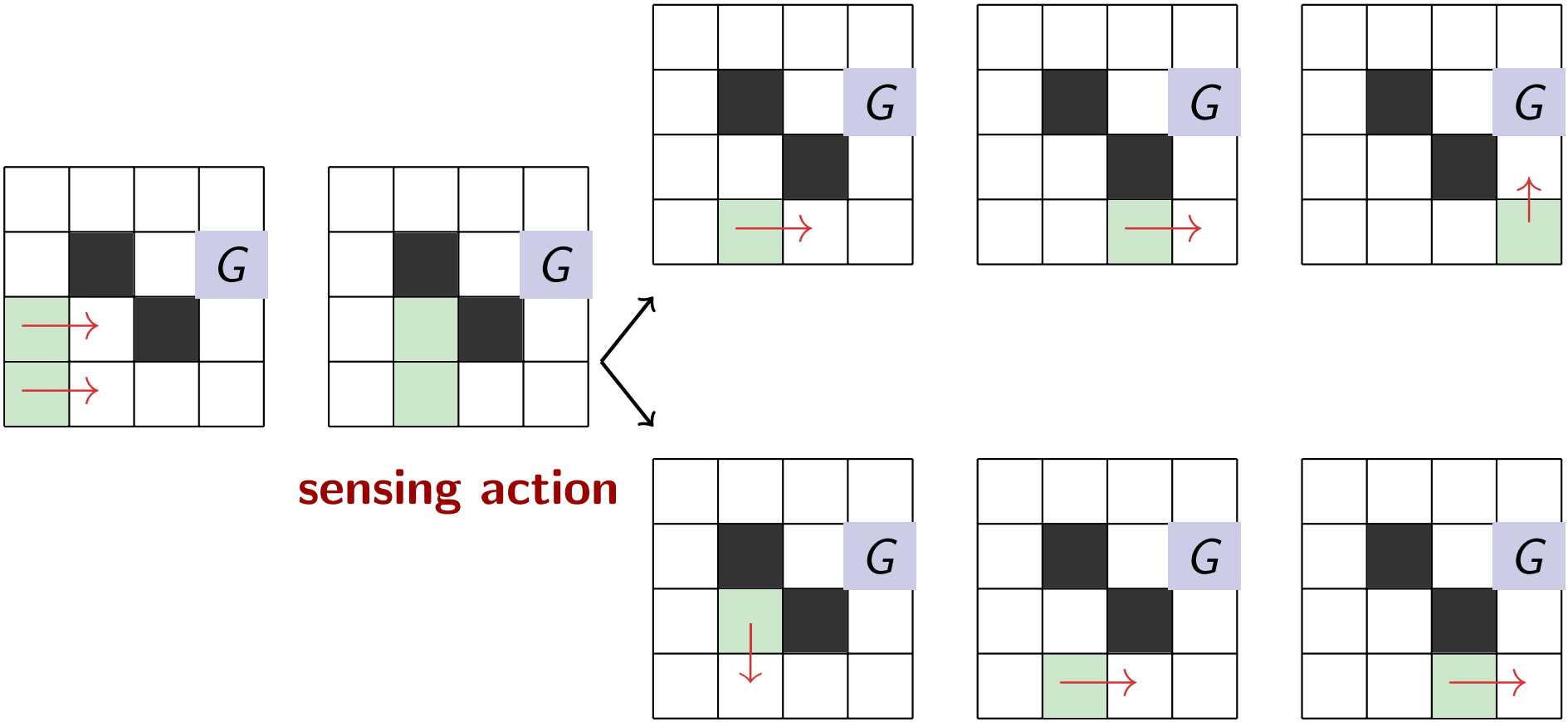
Contingent planning

- Can be seen as action planning with perception actions.
- The plan can be interpreted as an automaton, the execution of which controls the system
- by reading the output of the system (observations) and sequentially providing the input for the system (planned actions).
- Knowing how to acquire information is strongly related to solving contingency planning problems, as this information is likely to reduce the size of the belief state $\#b$.

$\rightarrow$ **After the application of a sensing action, only states compatible with the perceived observation are possible.**

Contingent planning

Example with deterministic transitions: Maze Problem



Contigent planning

- when a plan is computed offline, **the plan is in the form of a tree (conditional plan)**
- a tree **branching off into possible states depending on the result of perception**
- a graphic can be considered instead, when repeated states are considered.
- This task of **planning under uncertainty** can be formulated as a problem of **searching in the space of belief states**.
- **Difficulties:** 1) derivation of a heuristic function to guide the search, 2) the belief state representation and update.

input : $\mathcal{M} = \langle S, b_0, S_G, A, O, \varphi, \omega \rangle$
output: Plan π
return OR_search(b_0, [ ])

Procedure *OR_search*(n,π)
- **if** *is_goal*(n) **then**
  - **return** [ ]
- **foreach** $a \in A$ **do**
  - $n' \leftarrow$ next(n, a)
  - $\pi \leftarrow$ AND_search(n', [ ])
  - **if** $\pi \neq$ fail **then**
    - **return** $\pi \circ a$
- **return** fail

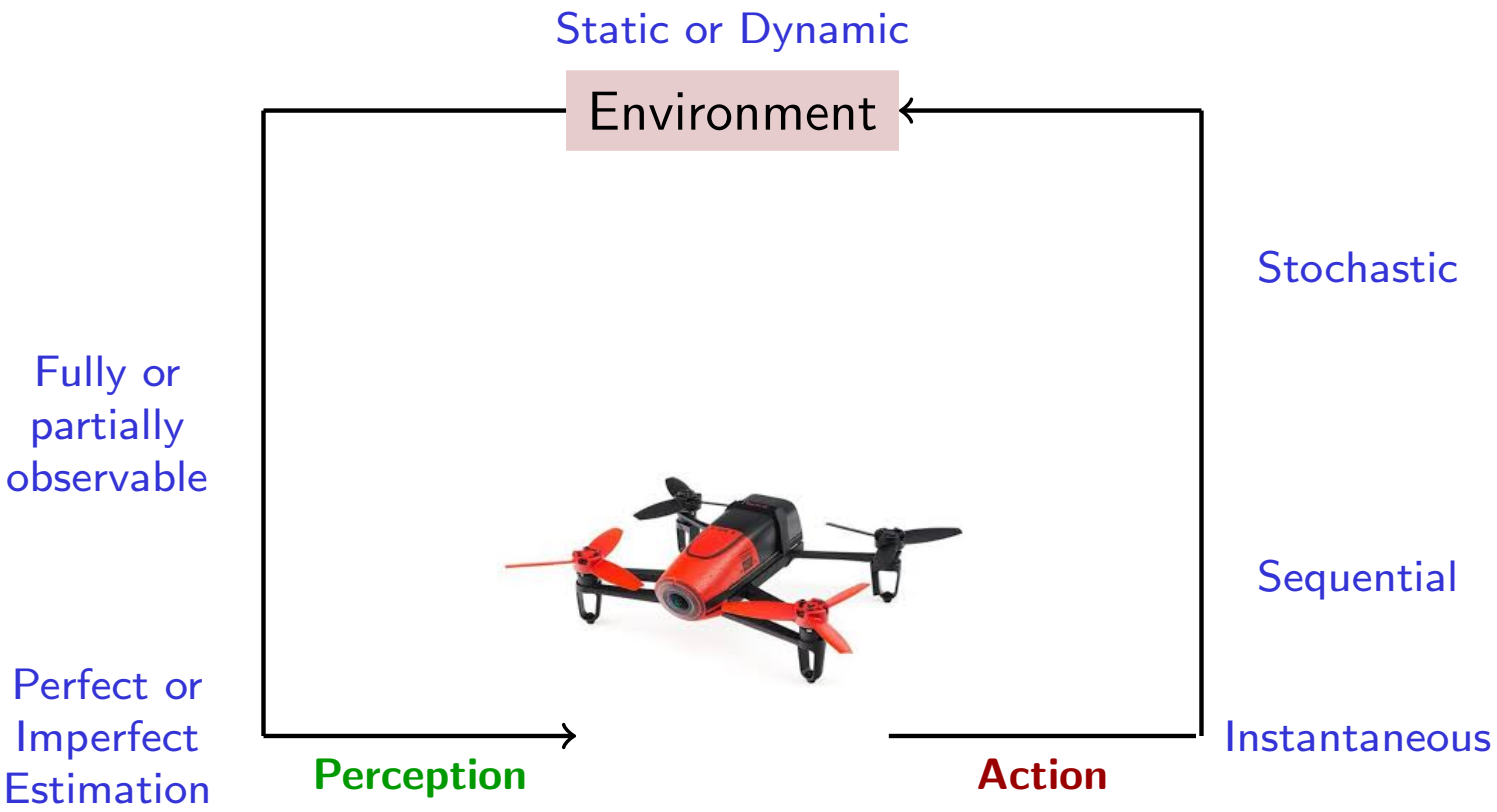
Procedure *AND_search*(n,π)
- **foreach** $o_i \in O$ **do**
  - $n_i \leftarrow$ next(n, o_i)
  - $\pi_i \leftarrow$ OR_search(n_i, [ ])
  - **if** $\pi_i =$ fail **then**
    - **return** fail
- **return** $\left[\textit{if } o_i \textit{ then } \pi_i \right]_i$

This formulation is the basis for the most recent conformant/contingent planners such as Conformant-FF (Brafman and Hoffmann, 2004), MBP (Bertoli et al., 2006), POND (Bryce et al., 2006), and T1 (Albore et al., 2011).

Outline

Probabilistic planning

Assumptions:



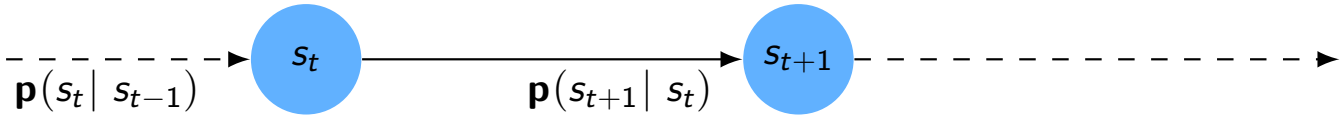
$\rightarrow$ **uncertainty on the initial state**

Markov Chains

- Dynamic system with discrete time steps $(s_t)_{t \in N} \in S$, where S is the finite state space such that:

$$\mathbb{P}(s_{t+1} = s | s_t, s_{t-1}, ..., s_0) = \mathbb{P}(s_{t+1} = s | s_t)$$

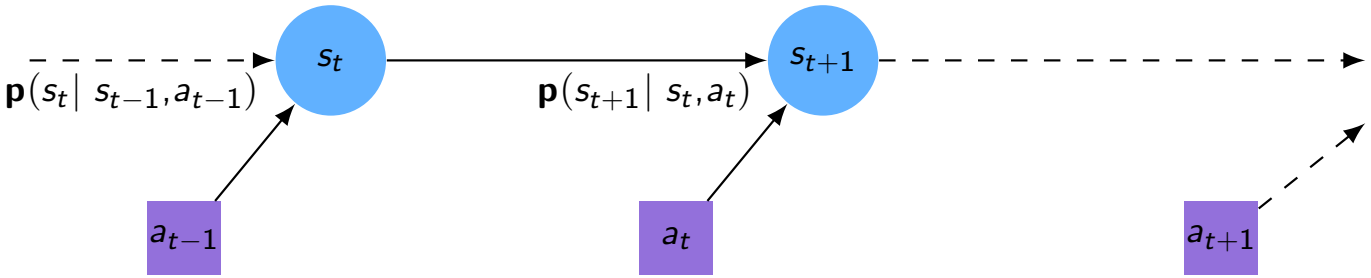
All information that is relevant to the prediction of the future is contained in the present state (property of Markov).



- Thus a Markov Chain on S is defined by an initial state s_0 (or more generally by $\mathbb{P}(s_0)$) and the transition probability values: $p(y|x) = \mathbb{P}(s_{t+1} = y | s_t = x)$.

Markov Decision Process (MDP)

- A controlled Markov Chain see as a *Markov Decision Process*



- A fundamental tool for probabilistic planning.
- History:
  - originated in the 1950s: first works by Bellman and Howard (in connection with the optimal control of discrete systems)
  - in the 50s until the 80s: theory, first algorithms and applications
  - From the 1990s onwards: MDPs appear in AI literature
- **MDP from an AI perspective**
  - **Probabilistic Planning**
  - Reinforcement Learning

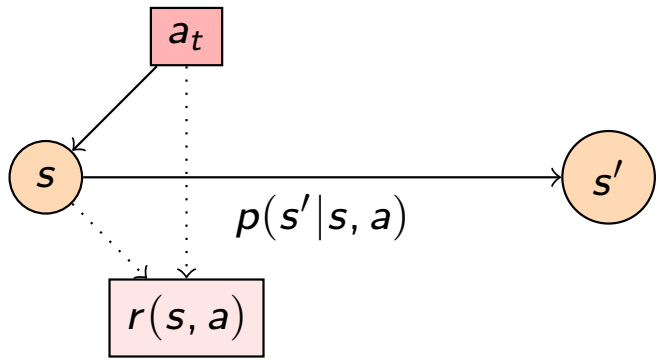
Markov Decision Process (MDP)

What are MDPs good for?

- Dynamic and uncertain domains
- Sequential Decision
- Rational Decision
- Domains with cycles
- Full observability ("perfect" sensors)

- One kind of uncertainty: effects of the actions of the autonomous agent
- **Goal :** Act by maximizing the expectation of accumulated rewards over the **long-term** (or minimizing the expectation of the long-term cost). For this, a policy defining this behaviour is computed, denoted by $\pi(s) : s \rightarrow a$

n-uplet:
$\{S, A, \mathcal{T}, R\}$



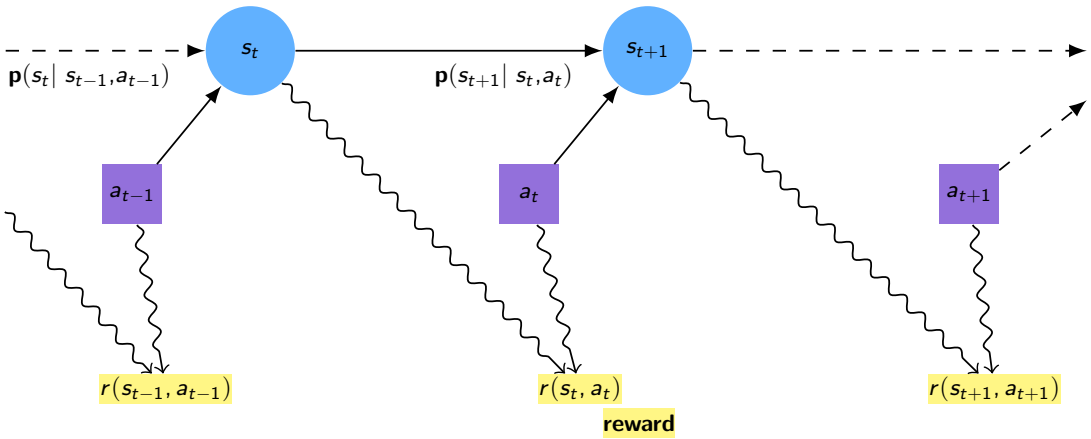
Markov Decision Process (MDP)

Suggested textbook: Planning with Markov Decision Processes: an AI perspective (Mausan and Kolobov, 2012)

Definition of the MDPs :

an MDP is a n-uplet $\{S, A, \mathcal{T}, R\}$ where:

- S is a finite space of states,
- A is a finite space of actions,
- $\mathcal{T} : S \times A \times S \rightarrow [0, 1]$ is a transition function specifying the probability value of the transition to the state $s' \in S$ from the state $s \in S$ and by using the action $a \in A$, such that $\mathcal{T}(s', a, s) = p(s'|s, a)$,
- $R : S \times A \rightarrow \mathbb{R}$ is a stationary reward function $r(s, a)$ that defines the rewards (or costs) when the agent uses the action $a \in A$ in the system state $s \in S$



Markov Decision Process (MDP)

finite-horizon non-stationary MDP (Mausan and Kolobov, 2012) :

such an MDP is a n-uplet $\{S, A, \mathcal{T}, R, D\}$ where:

- S is a finite set of states,
- A is a finite space of actions,
- **D is a finite sequence of integers $(1, 2, 3, ..., T_{max})$, representing the decision steps,**
- **$\mathcal{T} : S \times A \times S \times D \to [0, 1]$ is a transition function specifying the probability value of the transition to the state $s' \in S$ from a given state $s \in S$, using the action $a \in A$ and at the decision step t, such that $\mathcal{T}(s', a, s, t) = p(s'|s, a, t)$,**
- **$R : S \times A \times D \to \mathbb{R}$ is a reward function $r(s, a, t)$ that defines the rewards (or costs) when the action $a \in A$ is used in state $s \in S$ during decision step t.**

Markov Decision Process (MDP)

stationary infinite-horizon MDP (Mausan and Kolobov, 2012) :

MDP is a n-uplet $\{S, A, \mathcal{T}, R, \gamma\}$ where:

- S is a finite set of states,
- A is a finite set of actions,
- $\mathcal{T} : S \times A \times S \rightarrow [0, 1]$ is a stationary transition function that specifies the probability value of the transition to the state $s' \in S$ from a state $s \in S$ using action $a \in A$, such that $\mathcal{T}(s', a, s) = p(s'|s, a)$
- $R : S \times A \rightarrow \mathbb{R}$ is a stationary reward function $r(s, a)$ that defines the rewards (or costs) when action $a \in A$ is used in state $s \in S$,
- $\gamma \rightarrow [0, 1[$ **is the discount factor**

Markov Decision Process (MDP)

stationary infinite-horizon MDP (Mausan and Kolobov, 2012) :

MDP is a n-uplet $\{S, A, \mathcal{T}, R, \gamma\}$ where:

- S is a finite set of states,
- A is a finite set of actions,
- $\mathcal{T} : S \times A \times S \rightarrow [0, 1]$ is a stationary transition function that specifies the probability value of the transition to the state $s' \in S$ from a state $s \in S$ using action $a \in A$, such that $\mathcal{T}(s', a, s) = p(s'|s, a)$
- $R : S \times A \rightarrow \mathbb{R}$ is a stationary reward function $r(s, a)$ that defines the rewards (or costs) when action $a \in A$ is used in state $s \in S$,
- $\gamma \rightarrow [0, 1[$ **is the discount factor**

Whether it's a finite or infinite horizon MDP ...

- The objective is to **control the system** by choosing the appropriate actions to optimize a given criterion, for example, **maximizing the total expected gain**.
- **The general form of a solution of an MDP is a policy depending on the history of the process** $h_n = (s_1, a_1, s_2, a_2, ..., s_n, a_n) \in H$. Given a history, this policy returns a probability distribution on actions, $\pi : H \times A \rightarrow [0, 1]$.

Policy

Deterministic policy

A **deterministic policy** is a deterministic distribution over actions whatever the history, *i.e.* for each history, there exists an action with probability 1. Such a policy can be written $\pi : H \to A$.

- When the initial state (or a probability distribution on the initial state) is known, any random history-dependent policy can be replaced by a random **Markov policy** with the same value function.

Markov policy

A Markov policy $\pi : S \times D \times A \to [0, 1]$ only depends on the current state $s \in S$ and on the decision step $t \in D$ in the finite horizon case (resp. $\pi : S \times A \to [0, 1]$ for the infinite horizon case)

- **Les politiques Markoviennes peuvent être évaluées par des fonctions de valeurs Markoviennes**

Optimal policy and value function

- A **utility function (or value function) evaluates a policy** for a given history h_n: $V^\pi(h_n) = utility(\pi, R_n, R_{n+1}, ...R_T)$ where $(R_T)_{T \geqslant n}$ are the random variables representing the future rewards.

Let us assume a **rational** agent that is **risk-neutral**, *i.e.*

- **the utility of a policy is defined as the expectation of the sum of the future rewards:**

$$V^\pi(h) = \mathbb{E}\left[\sum_{t=0}^{T} r(s_t, \pi(s_t)) \mid h\right]$$

or, in the infinite horizon case:

$$V^\pi(h) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid h\right]$$

- **Two policies that have the same expectation of gain have the same value.**
- **Thus the policy π^* is optimal if $V^{\pi^*}(h) \geq V^\pi(h), \forall \pi \forall h$**

Existence of an optimal solution

Guarantees on the existence of an optimal policy

If the quality of every policy can be measured by its expected linear additive utility, there is a policy that is optimal at every time step (Bellman, 1954)

- **A policy is optimal if its value function is not lower that any other value function**:
  $\rightarrow$ however, π^* is not necessarily unique and may not even exist.
  $\rightarrow$ A particular MDP may have multiple distinct optimal policies.
- Because of the Markov property, it can be shown that the optimal policy is a function of the current state
  If V^* exists and is Markovian : π^* exists and is a deterministic Markov policy (Mausan and Kolobov, 2012).

MDP solving

To solve an infinite horizon MDP it suffices to look for a stationary deterministic Markov policy π that maximizes the value function defined as:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s\right]$$

We can write it:

$$V^{\pi}(s_0) = \mathbb{E}\left[\gamma^0 r(s_0, \pi(s_0))\right] + \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_t = s_1, s_0, \pi(s_0)\right]$$

$$V^{\pi}(s_0) = r(s_0, \pi(s_0)) + \gamma \sum_{s \in S} p(s|s_0, \pi(s_0)) V^{\pi}(s)$$

The optimal (stationary deterministic Markov) policy π^* can be computed along with the optimal value function $V^* = V^{\pi^*}$, based on the Bellman equation:

$$V^*(s) = \max_{a \in A}\left[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')\right]$$

$$\pi^*(s) = \arg\max_{a \in A}\left[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')\right]$$

Bellman operator

The Bellman operator - stationary MDP and discounted criterion case

Let's define $LV(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{(}s') \right]$

- an application of the operator is guarantee to improve the value of the state ($\geq$) being treated.
- the operator L is a contraction operator for V in the value function space:

$$||LV - LU|| = \max_s |LV(s) - LU(s)| \leq \gamma ||V - U||, \forall V, U \in \mathbb{V}, \forall s \in S$$

- when $LV_\pi = V_\pi$, the Value function is said to be converged to a fixed-point.

$\rightarrow$ since the Bellman operator is a contraction in the value function space, we can apply **Linear Programming** to solve an MDP, such as:

$$\min_{V \in \mathcal{V}} \sum_{s \in S} V(s) \text{ with: } V(s) \geq [r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')]$$

then, for each $s \in S$:

$$\pi(s) = \arg\max_{a \in A} [r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')]$$

Optimality principle

Suggested textbook: Dynamic Programming and Optimal Control (Bertsekas, 1995)

Principle of optimality

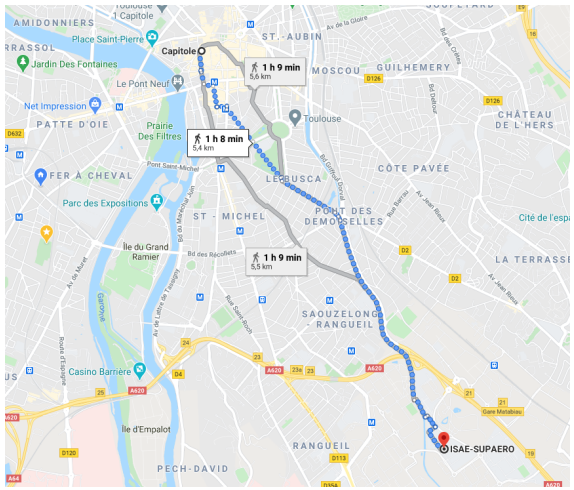
Let $\pi^* = \{\pi_0^*, \pi_1^*, ..., \pi_{N-1}^*\}$ be an optimal policy for the problem, and assume that when using π^* a given state s_i occurs with positive probability. Consider the subproblem whereby one is at s_i at time step $i < N$, and wish to maximize the expected rewards from time i until N:

$$V^{\pi_i^*}(s_i) = \mathbb{E}\left[\sum_{t=i}^{N} r(s_t, \pi_i^*(s_t)) \mid s_i\right]$$

then, the truncated policy $\{\pi_i^*, ..., \pi_{N-1}^*\}$ is optimal for this subproblem.

The intuition behind this principle is that:

- suppose shortest path problem to go from SUPAERO to Capitole
- if the shortest path between SUPAERO and Capitole passes throw Pont de Demoiselles
- the principle translates to the fact that the Pont de Demoiselles portion is also the shortest route between Pont de Demoiselles and Capitole

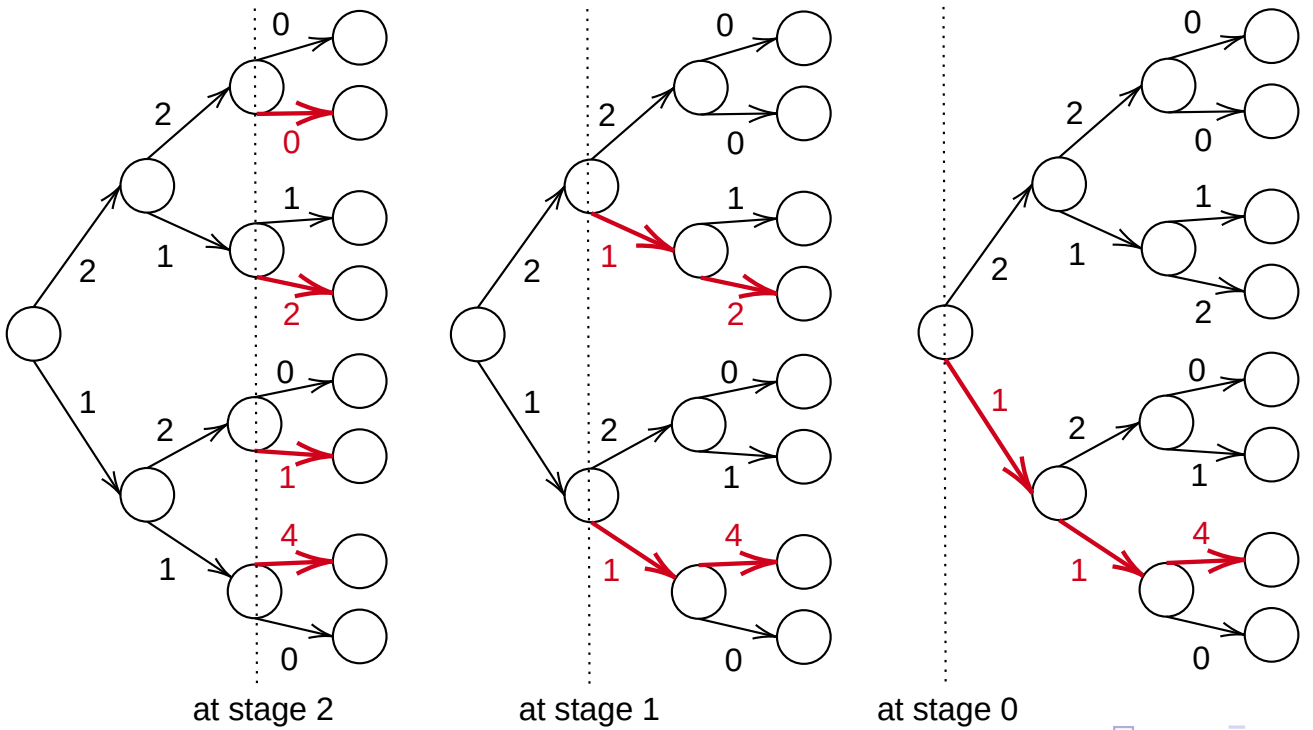


Dynamic Programming

The principle suggests that an optimal policy can be constructed in piecemeal fashion :

- first constructing an optimal policy for the sub-problem involving the last stage,
- then extending the optimal policy to the sub-problem involving the last two stages,
- and continuing in this manner until the an optimal policy for the entire problem is found.

Dynamic Programming algorithm: it proceeds sequentially, by solving all the sub-problems of a given time length, using the solution of the sub-problem of shorter time length.



Solving algorithms

A range of algorithms for exact or approximate MDP resolution, almost based on DP exist:

- Fundamental Algorithms
  $\rightarrow$ **value iteration** (Gauss-Seidel, assynchrone,...), **policy iteration** (Howard), Linear Programming
- Heuristic Search Algorithms
  $\rightarrow$ LAO* (Hansen and Zilberstein, 2001) and extensions, **RTDP** (Barto et al., 1995) and extensions, Glutton (Kolobov et al., 2012), Gourmand (Kolobov et al., 2016),
- Sampling based techniques
  $\rightarrow$ **UCT** (Kocsis and Szepesvári, 2006), PROST (Keller and Eyerich, 2012), THTS (Keller and Helmert, 2013)
- Determinization-based techniques
  $\rightarrow$ FF-Replan (Yoon et al., 2007) , RFF (Teichteil-Königsbuch et al., 2010)
- and many others!

Value Iteration Algorithm

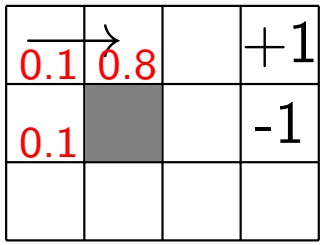
Algorithm 1: Value Iteration

Data: ϵ and $\{S, A, \mathcal{T}, R, \gamma\}$
Result: V and π
$n \leftarrow 0$
initialiser $V_n(s) = \max_{a \in A} r(s, a)$
do
  forall $s \in S$ **do**
    $V_{n+1}(s) = \max_{a \in A} [r(s, a) + \gamma \sum_{s' \in S} p(s'|s_i, a) V_n(s')]$
  $n \leftarrow n + 1$
while $|V_{n+1} - V_n| > \epsilon$
forall $s \in S$ **do**
  $\pi(s) \leftarrow \arg\max_{a \in A} [r(s, a) + \gamma \sum_{s' \in S} p(s'|s_i, a) V_n(s')]$
return V_n, π

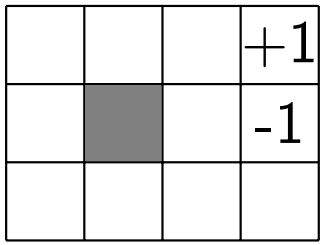
Example

The agent (robot) is on a grid;

- its goal is to reach the area $+1$, as soon as possible, avoiding -1 area;
- The walls are blocking the robot's path;
- The agent's actions don't always go as planned:
  $\rightarrow$ 80% of the time and if there's no wall, the agent ends up in the area east of its previous area at the end of the action *East*.
  $\rightarrow$ in 10% of the time and if there's no wall, the agent ends up in the south-side adjacent area ; Finally in 10% of the time, it remains in the previous area.
- Large rewards are at the end;



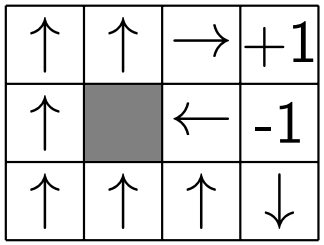
Ex. of action



Maze Problem

Example

Value Iteration Example:
First iteration :
$\vdots$
state s_3:
$V_1(s_3) = \max_{a \in A} [r(s_3, a) + \gamma \sum_{s' \in S} p(s'|s_3, a) V_0(s')]$
for action East:
$V_1(s_3) = [0 + 0.95(0.1 \times 0 + 0.1 \times 0 + 0.80 \times 1)]$
$V_1(s_3) = 0.76$
$\vdots$
state s_9:
$V_1(s_9) = \max_{a \in A} [r(s_9, a) + \gamma \sum_{s' \in S} p(s'|s_9, a) V_0(s')]$
for action North:
$V_1(s_9) = [0 + 0.95(0.1 \times 0 + 0.1 \times 0 + 0.80 \times 0)]$
$V_1(s_9) = 0$
$\vdots$



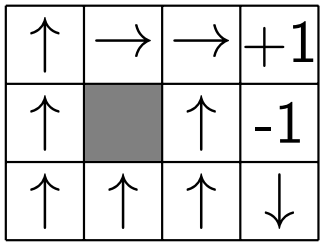
Policy of iter=1

0	0	0.76	1.95
0		0	-1.95
0	0	0	0

Value of iter=1

Example

Value Iteration Example:
second iteration:
$\vdots$
state s_3:
$V_2(s_3) = \max_{a \in A} [r(s_3, a) + \gamma \sum_{s' \in S} p(s'|s_3, a) V_1(s')]$
for action East:
$V_2(s_3) = [0 + 0.95(0.1 \times 0.76 + 0.1 \times 0 + 0.80 \times 1.95)]$
$V_2(s_3) = 1.55$
$\vdots$
state s_9:
$V_2(s_9) = \max_{a \in A} [r(s_9, a) + \gamma \sum_{s' \in S} p(s'|s_9, a) V_1(s')]$
for action North:
$V_2(s_9) = [0 + 0.95(0.1 \times 0 + 0.1 \times 0 + 0.80 \times 0)]$
$V_2(s_9) = 0$
$\vdots$



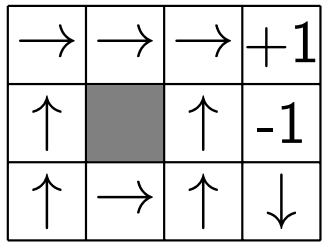
Policy of iter=2

0	0.57	1.55	2.85
0		0.39	-2.85
0	0	0	0

Value of iter=2

Example

Value Iteration Example:
fifth iteration:
$\vdots$
state s_3:
$V_5(s_1) = \max_{a \in A} [r(s_1, a) + \gamma \sum_{s' \in S} p(s'|s_1, a) V_4(s')]$
for action East:
$V_5(s_1) = [0 + 0.95(0.1 \times 3.13 + 0.1 \times 1.52 + 0.80 \times 4.52)]$
$V_5(s_1) = 3.88$
$\vdots$
state s_9:
$V_5(s_9) = \max_{a \in A} [r(s_9, a) + \gamma \sum_{s' \in S} p(s'|s_9, a) V_4(s')]$
for action East:
$V_5(s_9) = [0 + 0.95(0.2 \times 0.22 + 0.80 \times 0.72)]$
$V_5(s_9) = 0.59$
$\vdots$



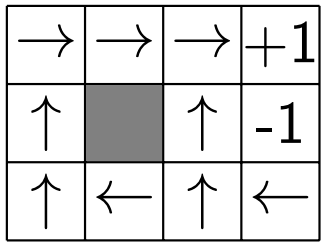
Policy of iter=5

1.67	2.76	3.88	5.29
0.84		2.09	-5.29
0.27	0.59	1.18	0.12

Value of iter=5

Example

Value Iteration Example:
29^{th} iteration:
$\vdots$
state s_3:
$V_{29}(s_1) = \max_{a \in A} [r(s_1, a) + \gamma \sum_{s' \in S} p(s'|s_1, a) V_{28}(s')]$
for action East:
$V_{29}(s_1) = [0 + 0.95(0.1 \times 13.18 + 0.1 \times 9.76 + 0.80 \times 15.48)]$
$V_{29}(s_1) = 14.0$
$\vdots$
state s_9:
$V_{29}(s_9) = \max_{a \in A} [r(s_9, a) + \gamma \sum_{s' \in S} p(s'|s_9, a) V_{28}(s')]$
for action West:
$V_{29}(s_9) = [0 + 0.95(0.2 \times 8.63 + 0.80 \times 9.49)]$
$V_{29}(s_9) = 8.85$
$\vdots$



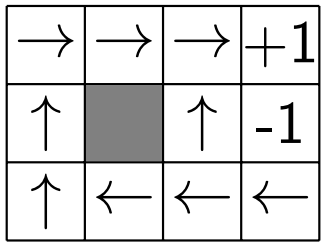
Policy of iter=29

11.7	12.8	14.0	15.7
10.7		9.9	-15.7
9.71	8.85	8.75	5.57

Value of iter=29

Example

Value Iteration Example:
160^{th} iteration:
$\vdots$
state s_3:
$V^*(s_1) = \max_{a \in A} [r(s_1, a) + \gamma \sum_{s' \in S} p(s'|s_1, a) V^*(s')]$
for action East:
$V^*(s_1) = [0 + 0.95(0.1 \times 18.1 + 0.1 \times 13.16 + 0.80 \times 19.99)]$
$V^*(s_1) = 18.1$
$\vdots$
state s_9:
$V^*(s_9) = \max_{a \in A} [r(s_9, a) + \gamma \sum_{s' \in S} p(s'|s_9, a) V^*(s')]$
for action West:
$V^*(s_9) = [0 + 0.95(0.2 \times 13.02 + 0.80 \times 13.88)]$
$V^*(s_9) = 13.0$
$\vdots$



Optimal Policy

15.8	17.0	18.1	19.9
14.9		13.1	-19.9
13.8	13.0	12.3	8.24

Optimal Value

Policy Iteration Algorithm

Algorithm 2: Policy Iteration

Data: $\{S, A, \mathcal{T}, R, \gamma\}$
Result: V and π
initialiser π_n, π_{n+1}
do
  $\pi_n \leftarrow \pi_{n+1}$
  forall $s \in S$ **do**
    $V_n(s) = r(s, \pi_n(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi_n(s)) V_n(s')$
  forall $s \in S$ **do**
    $\pi_{n+1}(s) \leftarrow \arg\max_{a \in A} [r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V_n(s')]$
while $\pi_n \neq \pi_{n+1}$
return V_n, π_n

Q-value definition

Q-value under a Value Function

The Q-value of state s and action a under a value function V is denoted as $Q^V(s,a)$, such as:

$$Q^V(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s')$$

It is the one-step lookahead computation of the value of taking a in s under the belief that V is the true expected value (e.g. cost to reach a goal)
It is not worth to state that :

$$V(s) = \max_{a \in A} Q^V(s,a)$$

Greedy-policy

A greedy policy π^V for a value function V is a policy that in every state uses an action greedy w.r.t V

$$\pi^V(s) = \arg\max_{a \in A} Q^V(s,a)$$

RTDP : *Real-time Dynamic Programming*

- RDTP-based algorithms operate by **simulating the current greedy policy** to sample **paths**, or trajectories, **through the state space**, and performing **Bellman backups** only on the states in those trajectories.
- The process of sampling a **path** is called a **trial**
- RTDP is guaranteed to **converge asymptotically to V^*** over the states in the domain of **an optimal (partial) policy** starting from s_0
- The RTDP original version (designed for online planning) has two main **weaknesses**:
  $\rightarrow$ the **lack** of a principled **termination condition**.
  $\rightarrow$ it does not provide **any mechanisms to detect** when it gets **near the optimal** value function or policy

---

input : $\mathcal{M} = \langle S, s_0, G, A, T, R \rangle$
output: Policy $\hat{\pi}$
return *RDTP*(s_0)

Procedure *RTDP*(s_0)
  $V^l \leftarrow h$:a `heuristic value`
  while *there is time left* **do**
    RTDPtrial(s_0)
  return $\hat{\pi}(s_0)$

Procedure *RTDPtrial*(s_0)
  $s \leftarrow s_0$
  while $s \notin G$ **do**
    $a_{best} \leftarrow \arg\max_{a \in A} Q(s, a)$
    $V^l(s) \leftarrow Q(s, a_{best})$
    $s \leftarrow$`pick next state from` $T(s, a_{best}, s') \sim p(s'|s, a_{best})$
  return

Remark: trials can be performed until a goal state in G or until a given horizon

LRTDP : *Labeled Real-time Dynamic Programming*

- This is an improved version of RTDP proposed by (Bonet, 2003)
- It makes use off a solve-label mechanism called in the end of a **trial**
  $\rightarrow$ It defines a **residual**:
  $res(s) =\mid V(s) - \max_a(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V(s')) \mid$
  $\rightarrow$ and labels as **solved** the states for which $res(s) \leq \epsilon$
- In the **checkSolved** procedure, it performs value update again. It favors value back (ancestors) propagation.
- It keeps track of all the states which have already converged (labeled) and avoid visiting them again
- *provided that the goal state is reachable from every state and the initial value function is admissible and monotonic, then LRTDP solves a goal-oriented MDP in a number of trials bounded by:* $\frac{1}{\epsilon} \sum_{s \in S} V^*(s) - h(s)$

input : $\mathcal{M} = \langle S, s_0, G, A, T, R, \gamma, \epsilon \rangle$
output: Policy $\hat{\pi}$
return *LRDTP*(s_0)

Procedure *LRTDP*(s_0)
- $V^l \leftarrow h$:a `heuristic value`
- **while** *there is time left or* s_0 *not solved* **do**
  - *LRTDPtrial*$(s_0,)$
- **return** $\hat{\pi}(s_0)$

Procedure *LRTDPtrial*$(s_0,)$
- $s \leftarrow s_0$
- *visited* $\leftarrow []$
- **while** *s not solved* **do**
  - **if** *s not in visited* **then**
    - *visited.append(s)*
  - **if** $s \in G$ **then**
    - **break**
  - $a_{best} \leftarrow \arg\max_{a \in A} Q(s,a)$
  - $V^l(s) \leftarrow Q(s, a_{best})$
  - $s \leftarrow$`pick next state from` $T(s, a_{best}, s') \sim p(s'|s, a_{best})$
- **while** *visited not empty* **do**
  - $s = visited.pop(s_{last})$
  - **if** *not checkSolved*(s, γ, ϵ) **then**
    - **break**
- **return**

Rollout methods - sampling based techniques

The expression *rollout* has its origin in the work of (Tesauro and Galperin, 1997).

- it refers to rolling the dice in a game
- in that work a given position of the game is evaluated *rolling out* many games and averaging the scores obtained starting from this position

- Several recent algorithms approach the optimal solution by making use of rollout approximations:
  $\rightarrow$ following a one-step lookahead base policy or a multi-step lookahead base policy
- Example are: Monte-Carlo Tree Search (Browne et al., 2012), UCT and variants (Kocsis and Szepesvári, 2006; Keller and Helmert, 2013), PROST (Keller and Eyerich, 2012)

Monte-Carlo Tree Search

- A **tree is built** in an incremental manner.
- For each iteration of the algorithm, **a given tree policy/strategy** is used to find the most urgent node of the current tree.
- This **strategy** attempts to **balance exploration** (look in areas that have not been well sampled yet) and **exploitation** (look in areas which appear to be promising).
- A **simulation/rollout** is then run from the selected node and the search tree updated according to the result.
  $\rightarrow$ choices are made during this simulation according to some default policy, which in the simplest case is random
- This involves the **addition of a child node** corresponding to the action taken from the selected node, and an **update of the statistics** of its ancestors.

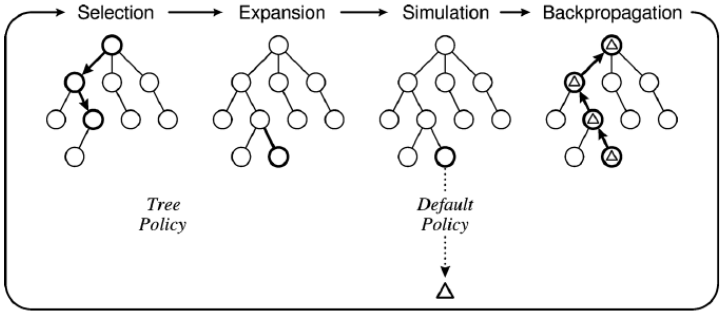


Figure from (Browne et al., 2012)

- **Benefits:** (1) values of intermediate states do not have to be evaluated; (2) Only the value of the terminal state at the end of each simulation is required.
- **Convergence Weaknesses :** (1) Q-value update based on all visited states generates a bias on the value; (2) convergence strongly depends on action selection exploit-explore strategy;

UCT: UCB applied to Monte-Carlo Tree Search

- **selectAction(s,d)**: implements **Upper Confidence Bounds (UCB1)** action selection strategy (Auer et al., 2002) that succeeds in resolving the **exploration-exploitation** tradeoff

$$a_{best} = \arg\max_{a \in A} Q(s,a) + c\sqrt{\frac{\log N_s}{N_{s,a}}}$$

$\rightarrow$ it modify the estimate of Q in order to forces the algorithm to explore actions that seem less promising.

- **Evaluate(s)**: implements a **rollout** simulation, usually proceeding a **multi-step** simulation with **random action selection**

input : $\mathcal{M} = \langle S, s_0, G, A, T, R \rangle$

Procedure *UCT*(s_0)
  while *there is time left* **do**
    UCTsearch(s_0)
  return $\hat{\pi}(s_0)$

Procedure *UCTsearch*(s,d)
  if s *is a Leaf node* **then**
    return *Evaluate*(s)
  $a_{best} \leftarrow$ selectAction(s)
  $s \leftarrow$ `pick next state from` $T(s, a_{best}, s') \sim p(s'|s, a_{best})$
  $q = r(s, a_{best}) + \gamma$UCTsearch($s'$,$d+1$)
  $N_s \leftarrow N_s + 1$
  $N_{s,a} \leftarrow N_{s,a} + 1$
  $Q(s, a_{best}) = Q(s, a_{best}) + \frac{q - Q(s, a_{best})}{N_{s,a}}$
  return q

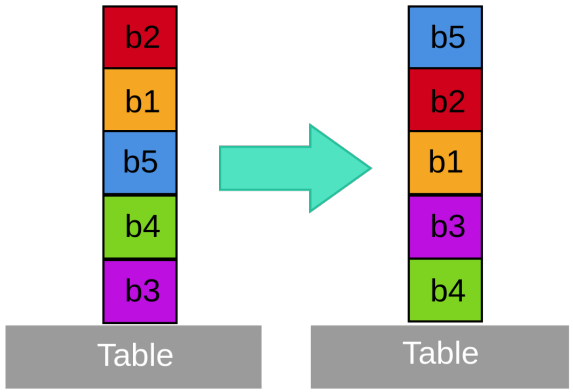
UCT as a MCTS-based method presents the same drawbacks

$\rightarrow$ Even if UCB drives nicely the search, it still suffers form the bias on the Q-value estimate.
$\rightarrow$ Recent UCT variants suggested improvements to these problems, see (Keller and Helmert, 2013)

Probabilistic Blocks world Example

```
(define (domain blocks-domain)
  (:requirements :probabilistic-effects :equality :typing)
  (:types block)
  (:predicates (emptyhand) (holding ?b - block) (on-table ?b - block) (on ?b1 ?b2 - block) (clear ?b - block))
  (:action unstack
    :parameters (?b1 ?b2 - block)
    :precondition (and (emptyhand) (not (= ?b1 ?b2)) (clear ?b1) (on ?b1 ?b2))
    :effect
      (probabilistic
        3/4 (and (holding ?b1) (not (emptyhand)) (clear ?b2) (not (clear ?b1)) (not (on ?b1 ?b2)))
        1/4 (and (clear ?b2) (emptyhand) (not (holding ?b1)) (clear ?b1) (on-table ?b1) (not (on ?b1 ?b2))))
  )
  (:action pickup
    :parameters (?b - block)
    :precondition (and (clear ?b) (on-table ?b) (emptyhand))
    :effect (probabilistic 3/4 (and (holding ?b) (not (emptyhand)) (not (on-table ?b)))
                           1/4 (and (on-table ?b) (emptyhand) (not (holding ?b)) ) )
  )
  (:action stack
    :parameters (?b1 ?b2 - block)
    :precondition (and (holding ?b1) (not (emptyhand)) (clear ?b2))
    :effect (probabilistic 3/4 (and (on ?b1 ?b2) (clear ?b1) (not (holding ?b1)) (emptyhand) (not (clear ?b2)))
                           1/4 (and (on-table ?b1) (clear ?b1) (emptyhand) (not (holding ?b1))))
  )
  (:action putdown
    :parameters (?b - block)
    :precondition (and (not (emptyhand)) (holding ?b))
    :effect (and (on-table ?b) (clear ?b) (emptyhand) (not (holding ?b)))
  )
)
```

```
(define (problem bw1_proba)
  (:domain blocks-domain)
  (:objects b1 b2 b3 b4 b5 - block)
  (:init (emptyhand) (on b1 b5) (on b2 b1) (on-table b3) (on b4 b3) (on b5 b4) (clear b2))
  (:goal (and (on b1 b3) (on b2 b1) (on b3 b4) (on-table b4) (on b5 b2) (clear b5)))
)
```



Demo

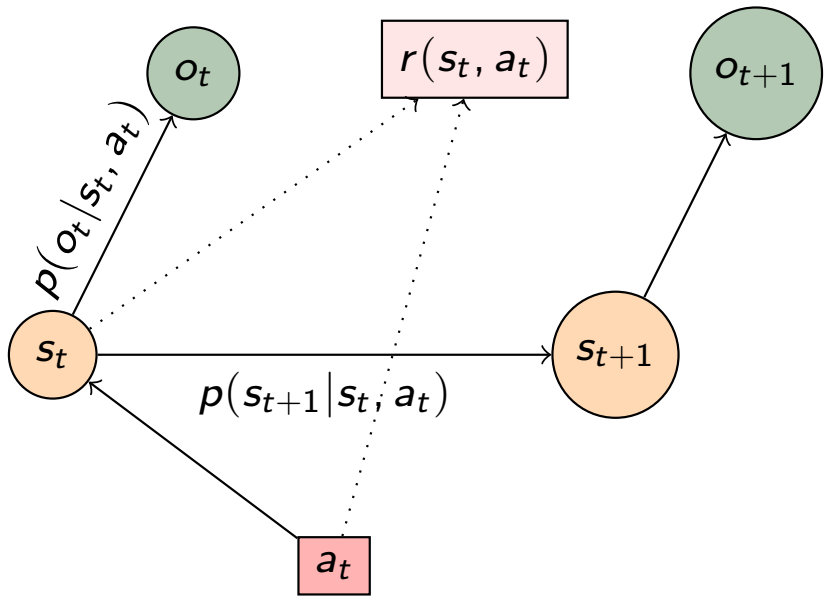
LRTDP algorithm implementation (and several others) from **Luis Pineda** Research Engineer, Facebook AI Research
https://github.com/luisenp/mdp-lib

And when observability is only partial

Partially Observable Markov Decision Processes

- Two kind of uncertainty:
  effects of agent actions
  state observations (agent + environment)

n-uplet:
$\{S, A, \Omega, p(), o(), r(), b_0\}$

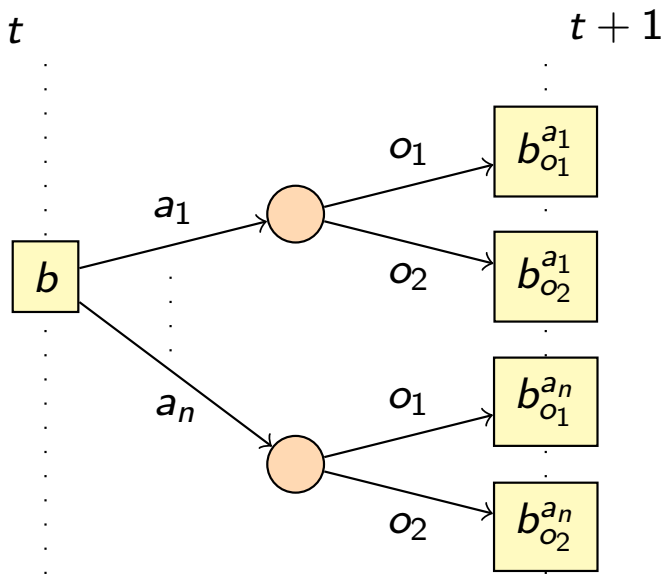


Transition of a POMDP.

And when observability is only partial

Partially Observable Markov Decision Processes

- Two kind of uncertainty:
  effects of agent actions
  state observations (agent + environment)
- state belief b:
  no direct access to s, thus a probability distribution over the states is updated



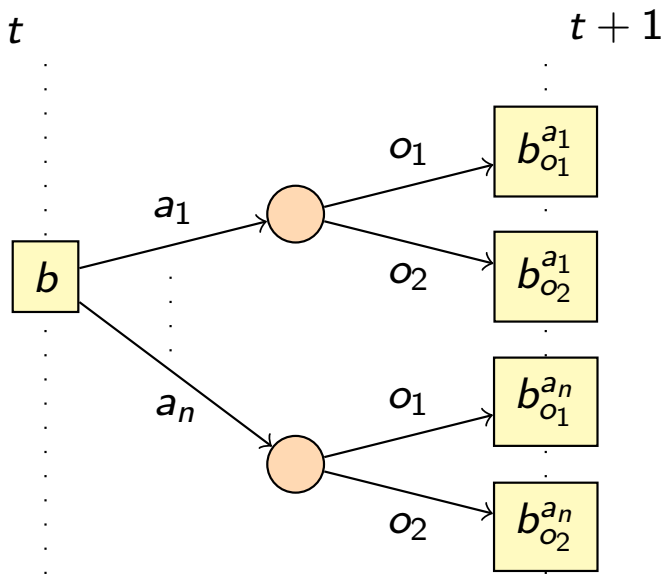
Belief update

$$b_a^o(s') = \frac{p(o|s',a)\sum_{s\in S} p(s'|s,a)b(s)}{\sum_{s\in S}\sum_{s'\in S} p(o|s',a)p(s'|s,a)b(s)}$$

And when observability is only partial

Partially Observable Markov Decision Processes

- Two kind of uncertainty:
  effects of agent actions
  state observations (agent + environment)
- state belief b:
  no direct access to s, thus a probability distribution over the states is updated
- **Goal:** act by maximizing the expectation of cumulative rewards over the **long term**



Belief update

$$b_a^o(s') = \frac{p(o|s',a)\sum_{s\in S} p(s'|s,a)b(s)}{\sum_{s\in S}\sum_{s'\in S} p(o|s',a)p(s'|s,a)b(s)}$$

POMDP principle

1. At each moment t, the agent does not have access to the current state $s \in S$, but maintains a belief state b_t (probability distribution over S).
2. When it applies an action $a \in A$, it modifies the state of the process with a probability $p(s' \mid s, a)$ of bringing it into the state s'.
3. Then, the agent perceives only the observation o which depends on the function $p(o \mid s', a)$. For each action applied, the agent receives a reward $r \in R$.

s_0: "tiger-left"
$p(o = HL|s_0) = 0.85$
$p(o = HR|s_0) = 0.15$

s_1: "tiger-right"
$p(o = HL|s_1) = 0.15$
$p(o = HR|s_1) = 0.85$

Actions:
- listening
- open left door
- open right door

Reward function:
- penalty for wrong opening: -100
- reward for correct opening: $+10$
- cost for listening action: -1

Observations:
- to hear the tiger from left door: HL
- to hear the tiger from right door: HR

Belief State

- A single observation does not directly identify the s state of the system : thus, **some form of memory is required**
- Let's define **the history of all observations and actions** performed as: $h_t := \{a_0, o_1, ..., o_{t-1}, a_{t-1}, o_t\}$
- The **size of this history can be very important** as a function of time.
- However, the history does not need to be represented explicitly for the selection of actions, it is **enough to rely on the current belief state**:

POMDP can be transformed into a belief state MDP

the agent synthesizes all available information from its past using a belief state $b(s)$, which is a posteriori probability distribution: $b_t(s) := Pr(s_t = s | o_t, a_{t-1}, o_{t-1}, ..., a_0)$
It can be updated each time an action is performed and on observation received:

$$
\begin{aligned}
b_o^a(s') &= p(s' \mid b, a, o) = \frac{p(o, s' \mid b, a)}{p(o \mid b, a)} \\
&= \frac{p(o \mid s', a) p(s' \mid b, a)}{p(o \mid b, a)} \\
&= \frac{p(o \mid s', a) p(s' \mid b, a)}{\sum_{s,s'} p(o \mid b, a, s, s') p(s, s' \mid b, a)} \\
&= \frac{p(o \mid s', a) \sum_{s \in S} p(s' \mid s, a) b(s)}{\sum_{s' \in S} p(o \mid s', a) \sum_{s \in S} p(s' \mid s, a) b(s)}
\end{aligned}
$$

Belief State

- **Belief states form a Markovian process:** the state of belief at $t+1$ depends only on the state of belief, action, and observation at t.
- In conclusion, the belief state at $t+1$ synthesizes all the information available up to the instant t.
- **The POMDP thus defined on belief states is difficult to solve** because b is defined on the continuous space of probability distributions over S.

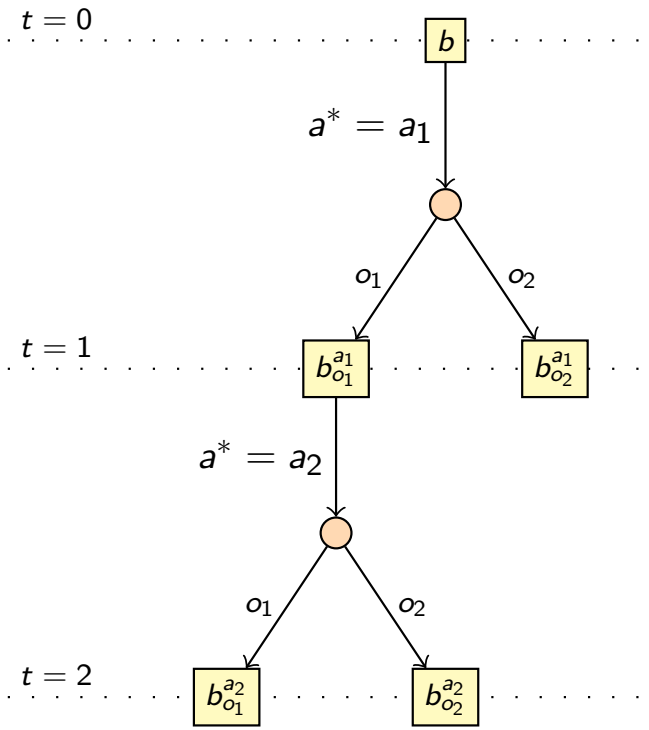
The agent's objective is to choose actions that will drive him to achieve better rewards.

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(b_t, \pi(b_t)) \middle| b_0 = b\right]$$

where, $0 \leqslant \gamma < 1$, and

$$r(b_t, \pi(b_t)) = \sum_{s \in S} r(s, \pi(b_t)) b_t(s)$$

Note a deterministic Markovian policy $\pi(b)$ is a function such as, $\pi(b) : b \mapsto a$.



POMDP solving - Maximizing rewards

Policy $\pi(b) : b \rightarrow a$

that maximizes the expectation of **the sum of discounted rewards** (γ-discounted criteron).

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \sum_{s} r(s, \pi(b_t)) b_t(s) \mid b_0 = b\right]$$

An optimal policy π^* is defined by V^*, that satisfies the Bellman equation:

$$V^*(b) = \max_{a \in A}\left[\sum_{s \in S} r(s,a) b(s) + \gamma \sum_{o \in \Omega} p(o|a,b) V^*(b_o^a)\right]$$

i.e. the solution is optimal when V^* converges to a fixed point for any b.

POMDP solving - Maximizing rewards

Policy $\pi(b) : b \rightarrow a$

that maximizes the expectation of **the sum of discounted rewards** (γ-discounted criteron).

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \sum_{s} r(s, \pi(b_t)) b_t(s) \mid b_0 = b\right]$$

An optimal policy π^* is defined by V^*, that satisfies the Bellman equation:

$$V^*(b) = \max_{a \in A}\left[\sum_{s \in S} r(s,a) b(s) + \gamma \sum_{o \in \Omega} p(o|a,b) V^*(b_o^a)\right]$$

i.e. the solution is optimal when V^* converges to a fixed point for any b.

- exact solution in infinite horizon settings is hard to compute:
  $\rightarrow$ PSPACE-complete (Papadimitriou and Tsitsiklis, 1987).

POMDP solving - Maximizing rewards

Policy $\pi(b) : b \rightarrow a$

that maximizes the expectation of **the sum of discounted rewards** (γ-discounted criteron).

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \sum_{s} r(s, \pi(b_t)) b_t(s) \mid b_0 = b\right]$$

An optimal policy π^* is defined by V^*, that satisfies the Bellman equation:

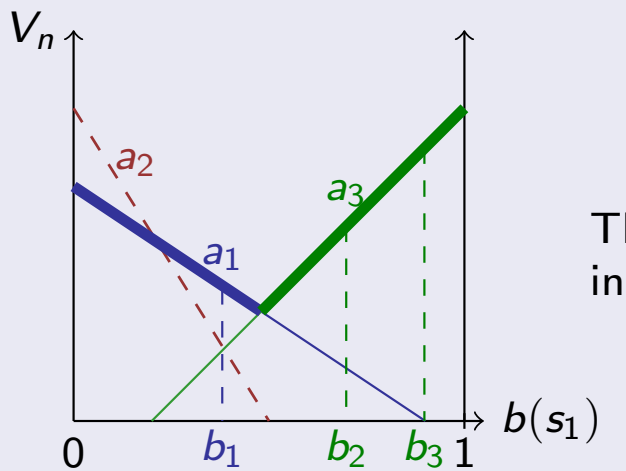
$$V^*(b) = \max_{a \in A}\left[\sum_{s \in S} r(s,a) b(s) + \gamma \sum_{o \in \Omega} p(o|a,b) V^*(b_o^a)\right]$$

i.e. the solution is optimal when V^* converges to a fixed point for any b.

- exact solution in infinite horizon settings is hard to compute:
  $\rightarrow$ PSPACE-complete (Papadimitriou and Tsitsiklis, 1987).
- researchers are interested in developing algorithms that approach the optimal solution, exploiting particular properties:
  $\rightarrow$ **of the value function;**
  $\rightarrow$ **of the system dynamics...**

Piece-wise Linear and Convex (PWLC) value function

- The optimal value function for a finite horizon problem is piece-wise linear and convex (PWLC), and can be approximated by a PWLC value function for an infinite horizon.
- This is due to the fact that the value function V_n can be parameterized by a finite set of vectors (hyperplanes) $\{\alpha_n^i\}, i = 1, ..., |V_n|$ (Cassandra, 1998).
- Each vector defines a region of the belief states space, where this vector represents the maximum value of V_n. The α-vectors form a partition of the belief state.
- The convexity and piece-wise linearity of the value function results from the fact that V_n realizes the maximum of the values of the actions, which are individually linear.



The value function of b at time step n can be defined by the existence of a set of α-vectors Γ_n such as :

$$V_n(b) = \max_{\alpha_n^i \in \Gamma_n} \sum_{s \in S} b(s)\alpha_n^i(s) \rightarrow V_n(b) = \max_{\alpha_n^i \in \Gamma_n} b \cdot \alpha_n^i$$

Thus, for a given b, the gradient of the value function is induced by the vector:

$$\alpha_n^b = \arg\max_{\alpha_n^i \in \Gamma_n} b \cdot \alpha_n^i$$

and, the policy $\pi(b)$ is the action associated with this vector $a(\alpha_n^b)$, such as : $\pi(b) = a(\alpha_n^b)$.

Exact value function iteration

- it consists on the iterative application of the dynamic programming operator.
- starting from an initial value approximation V_0 supposed PWLC
- the intermediate function $(V_1 \dots V_n)$ will also be PWLC.
- Algorithms manipulating α-vector have been proposed (Sondik, 1978).

They compute all possible projections LV_n, such as:

$$\begin{aligned} \Gamma^{a,*} \leftarrow \quad & \alpha^{a,*}(s) \quad = r(s,a) \\ \Gamma^{a,o} \leftarrow \quad & \alpha_i^{a,o}(s) \quad = \gamma \sum_{s' \in S} p(s'|s,a) p(o|s',a) \alpha_i(s'), \forall \alpha_i \in V_n. \end{aligned}$$

Thus, the cross-sum[a] is applied to complete the generation of the set of projections at $n+1$:

$$\Gamma^a = \Gamma^{a,*} \bigoplus \Gamma^{a,o_1} \bigoplus \Gamma^{a,o_2} \bigoplus \cdots \bigoplus \Gamma^{a,o_{|\Omega|}}$$

Finally, we can get the union over all sets Γ^a:

$$V_{n+1} \leftarrow \mathcal{L} V_n = \bigcup_a \Gamma^a$$

[a] the cross-sum of two sets of vectors $P = \{p_1, p_2, \dots, p_m\}$ et $Q = \{q_1, q_2, \dots, q_k\}$ is defined as: $P \oplus Q = \{p_1 + q_1, p_1 + q_2, \dots, p_1 + q_k, \dots, p_m + q_1, p_m + q_2, \dots, p_m + q_k\}$.

Exact value function iteration example

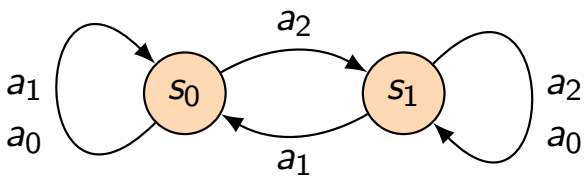
s_0: "tiger-left"
$p(o = HL|s_0) = 0.85$
$p(o = HR|s_0) = 0.15$

s_1: "tiger-right"
$p(o = HL|s_1) = 0.15$
$p(o = HR|s_1) = 0.85$

Actions:
- listening
- open left door
- open right door

Reward function:
- penalty for wrong opening: -100
- reward for correct opening: $+10$
- cost for listening action: -1

Observations:
- to hear the tiger from left door: HL
- to hear the tiger from right door: HR



rewards

R	a_0	a_1	a_2
s_0	-1.0	10	-100
s_1	-1.0	-100	10

listening (a_0)

T	s_1	s_2
s_0	1.0	0.0
s_1	0.0	1.0

O	s_0	s_1
o_0	0.85	0.15
o_1	0.15	0.85

open-left (a_1) or right (a_2)

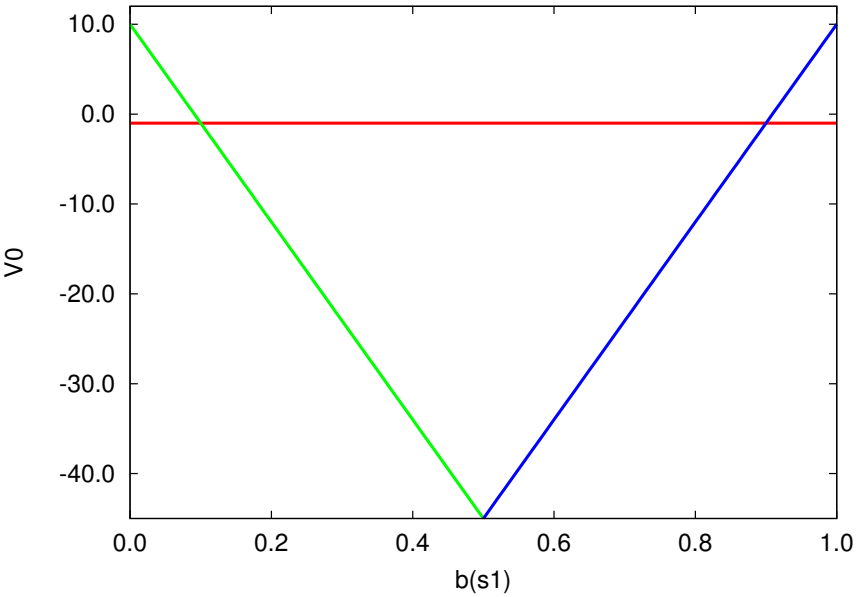
T	s_0	s_1
s_0	0.5	0.5
s_1	0.5	0.5

O	s_0	s_1
o_0	0.5	0.5
o_1	0.5	0.5

Exact value function iteration example

To start, let's define a set of α-vectors V_0 directly extracted from the reward function, i.e. an α-vector per action:

$$V_0 \leftarrow \alpha^a(s) = r(s,a), \forall a \in A$$



$$\Gamma^{a_0} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad \Gamma^{a_1} = \begin{bmatrix} 10 \\ -100 \end{bmatrix} \quad \Gamma^{a_2} = \begin{bmatrix} -100 \\ 10 \end{bmatrix}$$

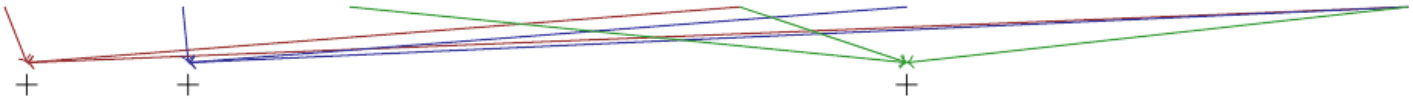
$$\bigcup_{a \in A}$$

$$V_0 = \left\{ \begin{bmatrix} -1 \\ -1 \end{bmatrix} \begin{bmatrix} 10 \\ -100 \end{bmatrix} \begin{bmatrix} -100 \\ 10 \end{bmatrix} \right\}$$

Exact value function iteration example

Considering a $\gamma = 0.95$, the projections for a first interaction can be processed:

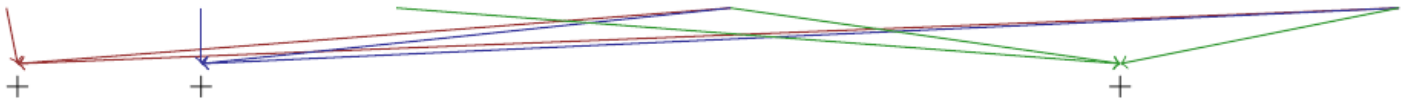
$$\Gamma_0^{a_0,o_0} = \left\{ \begin{bmatrix} -0.1425 \\ -0.8075 \end{bmatrix} \begin{bmatrix} 1.425 \\ -80.75 \end{bmatrix} \begin{bmatrix} -14.25 \\ 8.075 \end{bmatrix} \right\} \Gamma_0^{a_0,o_1} = \left\{ \begin{bmatrix} -0.8075 \\ -0.1425 \end{bmatrix} \begin{bmatrix} 8.075 \\ -14.25 \end{bmatrix} \begin{bmatrix} -80.75 \\ 1.425 \end{bmatrix} \right\} \Gamma^{a_0,*} = \left\{ \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\}$$



$$\Gamma^{a_0} = \left\{ \begin{bmatrix} -1.95 \\ -1.95 \end{bmatrix} \begin{bmatrix} 6.932 \\ -16.05 \end{bmatrix} \begin{bmatrix} -81.89 \\ -0.382 \end{bmatrix} \begin{bmatrix} -0.382 \\ -81.89 \end{bmatrix} \begin{bmatrix} 8.5 \\ -96 \end{bmatrix} \begin{bmatrix} -80.32 \\ -80-32 \end{bmatrix} \begin{bmatrix} -16.05 \\ 6.932 \end{bmatrix} \begin{bmatrix} -7.175 \\ -7.175 \end{bmatrix} \begin{bmatrix} -96 \\ 8.5 \end{bmatrix} \right\}$$

$$\vdots$$

$$\Gamma_0^{a_2,o_0} = \left\{ \begin{bmatrix} -0.475 \\ -0.475 \end{bmatrix} \begin{bmatrix} 4.75 \\ -47.5 \end{bmatrix} \begin{bmatrix} -47.5 \\ 4.75 \end{bmatrix} \right\} \Gamma_0^{a_2,o_1} = \left\{ \begin{bmatrix} -0.475 \\ -0.475 \end{bmatrix} \begin{bmatrix} 4.75 \\ -47.5 \end{bmatrix} \begin{bmatrix} -47.5 \\ 4.75 \end{bmatrix} \right\} \Gamma^{a_2,*} = \left\{ \begin{bmatrix} -100 \\ 10 \end{bmatrix} \right\}$$

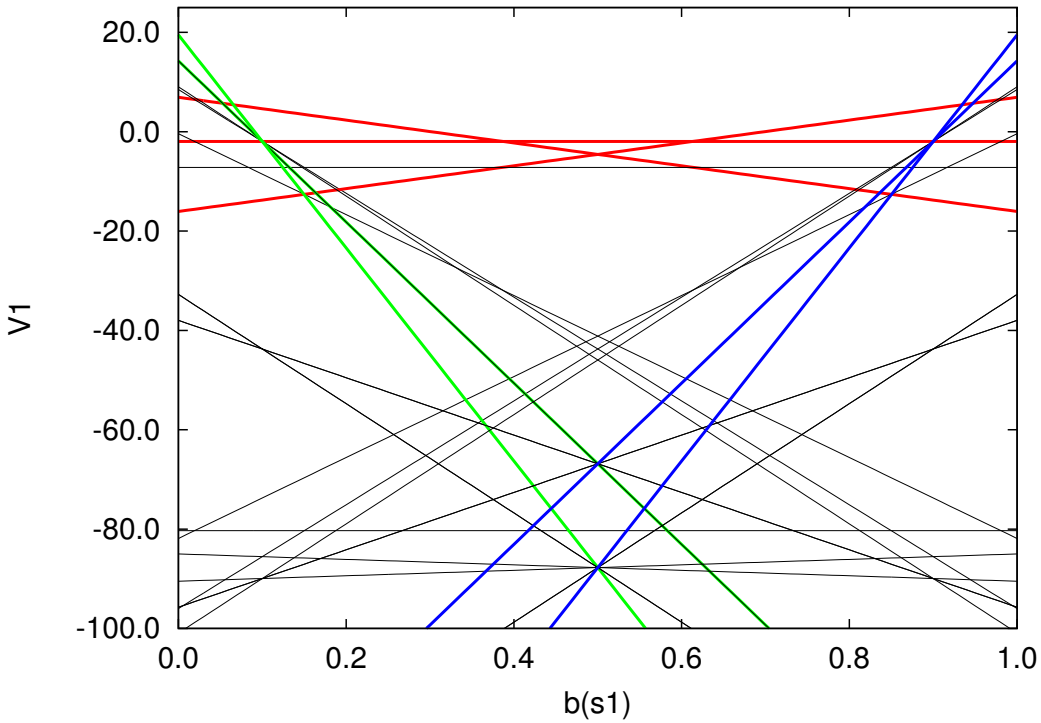


$$\Gamma^{a_2} = \left\{ \begin{bmatrix} -100.95 \\ 9.05 \end{bmatrix} \begin{bmatrix} -95.72 \\ -37.97 \end{bmatrix} \begin{bmatrix} -147.9 \\ 14.27 \end{bmatrix} \begin{bmatrix} -95.72 \\ -37.97 \end{bmatrix} \begin{bmatrix} -90.5 \\ -85 \end{bmatrix} \begin{bmatrix} -142.7 \\ -32.75 \end{bmatrix} \begin{bmatrix} -147.9 \\ 14.27 \end{bmatrix} \begin{bmatrix} -142.7 \\ -32.75 \end{bmatrix} \begin{bmatrix} -195 \\ 19.5 \end{bmatrix} \right\}$$

$$V_1 \leftarrow \bigcup_{a \in A} \Gamma^a = \left\{ \begin{bmatrix} -1.95 \\ -1.95 \end{bmatrix} \begin{bmatrix} -6.932 \\ -16.05 \end{bmatrix} \begin{bmatrix} -81.89 \\ -0.382 \end{bmatrix} \cdots \begin{bmatrix} -147.9 \\ 14.275 \end{bmatrix} \begin{bmatrix} -142.7 \\ -32.75 \end{bmatrix} \begin{bmatrix} -195 \\ 19.5 \end{bmatrix} \right\}$$

Exact value function iteration example

Resulting in:



- No need to go further to unsdertand that it exploses !!
- A number of α-vectors that is exponential in $|\Omega|$ is generated : $|V_{n+1}| = |A||V_n|^{|\Omega|}$ is generated at each iteration step.
- (Littman, 1997; Monahan, 1982) were interested in identifying unnecessary α-vectors and then ignoring them. This operation, called **pruning**, requires the resolution of a linear program for each α-vector.

The notion of reachable belief states and efficient algorithms

- some algorithms approximate the value function for a finite set of belief states $\mathcal{B}$ called **reachable belief states**.
- these algorithms are called **point-based algorithms**
  → **PBVI** (Pineau et al., 2003), Perseus (Spaan and Vlassis, 2004)
- others make use of a **trial-based** search guided by **different exploration heuristics**.
  → RTDP-bel (Bonet and Geffner, 2009), HSVI (Smith and Simmons, 2004, 2005), FSVI (Shani et al., 2007), SARSOP (Kurniawati et al., 2008), AEMS (Ross et al., 2008)
- and others, are based on **Monte-Carlo techniques** to approximate the value function with/without the need of belief state update operation
  → Monte Carlo Sampling (McAllester and Singh, 1999), Rollout (Bertsekas and Castanon, 1999), POMCP (Silver and Veness, 2010)

Offline algorithms

→ **approximate algorithms:**
QMDP (Littman et al., 1995),
PBVI (Pineau et al., 2003),
RTDP-bel (Bonet and Geffner, 2009),
Perseus (Spaan and Vlassis, 2004),
HSVI (Smith and Simmons, 2004, 2005),
FSVI (Shani et al., 2007),
SARSOP (Kurniawati et al., 2008).

Online algorithms

→ **approximate algorithms:**
RTBSS (Paquet et al., 2006),
Monte Carlo Sampling (McAllester and Singh, 1999),
Rollout (Bertsekas and Castanon, 1999),
Satia and Lave (Satia and Lave, 1973),
BI-POMDP (Washington, 1997),
AEMS (Ross et al., 2008),
POMCP (Silver and Veness, 2010)
DESPOT (Somani et al., 2013)

Approximate value iteration

Let's define a V_n parameterized by a set Γ_n of α-vectors and a given belief state $b \in \mathcal{B}$. One can compute α^b_{n+1} of V_{n+1} from LV_n (the unknown set of vectors):

$$\alpha^b_{n+1} = \underset{\alpha^i_{n+1} \in \mathcal{L}V_n}{\arg\max}\ b \cdot \alpha^i_{n+1},$$

Backup operator

Let's define $r_a = r(s,a)$ and $b \cdot r_a = \sum_s b(s) r(s,a)$, we have:

$$\begin{aligned}
V_{n+1}(b) = & \quad \max_a \left[b \cdot r_a + \gamma \sum_o p(o \mid a, b) V_n(b^a_o) \right] \\
= & \quad \max_a \left[b \cdot r_a + \gamma \sum_o p(o \mid b, a) \max_{\alpha^i_n \in V_n} \sum_{s'} b^a_o(s') \alpha^i_n(s') \right] \\
= & \quad \max_a \left[b \cdot r_a + \gamma \sum_o p(o \mid b, a) \max_{\alpha^i_n \in V_n} \frac{\sum_{s'} p(o|s',a) \sum_s p(s'|s,a) b(s) \alpha^i_n(s')}{p(o|b,a)} \right] \\
= & \quad \max_a \left[b \cdot r_a + \gamma \sum_o \max_{\alpha^i_n \in V_n} \sum_s \sum_{s'} p(o \mid s', a) p(s' \mid s, a) b(s) \alpha^i_n(s') \right]
\end{aligned}$$

thus: $V_{n+1}(b) = \max_a \left[b \cdot r_a + \gamma \sum_o \max_{\alpha^{a,o}_i \in \Gamma^{a,o}} b \cdot \alpha^{a,o}_i \right]$

Then, applying $\max_j b \cdot \alpha_j = b \cdot \arg\max_j b \cdot \alpha_j$ two times, we have:

$$backup(b) = \underset{\alpha^a_b \in \Gamma^a_b}{\arg\max}\, b \cdot \alpha^a_b, \text{ avec } \Gamma^a_b \leftarrow r_a + \gamma \sum_o \underset{\alpha^{a,o}_i \in \Gamma^{a,o}}{\arg\max}\, b \cdot \alpha^{a,o}_i$$

PBVI: Point-based Value Iteration (Pineau et al., 2003)

- This algorithm is one of the first performing algorithms that focuses on the construction of α-vectors and on the value function update for a finite set $\mathcal{B}$
- The set $\mathcal{B}$ is expanded at each iteration until a given size $|\mathcal{B}| \leq N_{\mathcal{B}}$.
- The update operator $\tilde{L}_{\text{PBVI}}$ calculates at each step the value function of the set $\mathcal{B}$, such as:

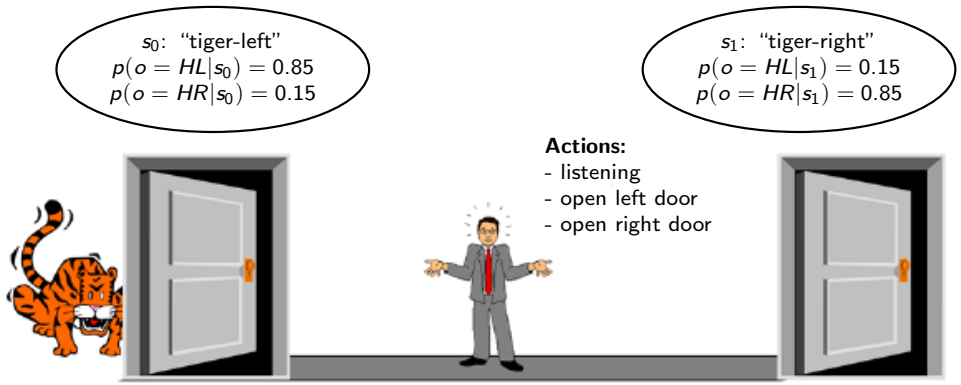
$$\tilde{L}_{\text{PBVI}} V_n = \bigcup_{b \in \mathcal{B}} backup(b)$$

- (Pineau et al., 2003) demonstrated that the error of the approximation of the value function is bounded and depends on the density of $\mathcal{B}$.

---

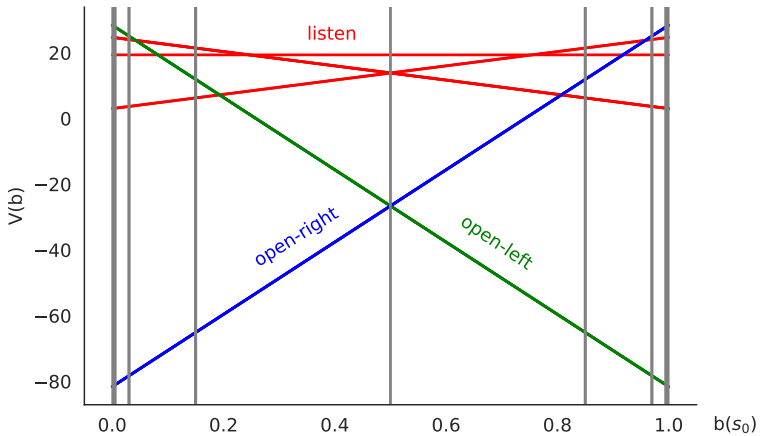
input : POMDP, N_i, $N_{\mathcal{B}}$
output: value function V
Initialize V_0, $n = 0$;
$\mathcal{B} \leftarrow b_0$;
$V_1 \leftarrow \emptyset$;
repeat
  $n = n + 1$;
  Expand $\mathcal{B}$;
  $V_n \leftarrow \emptyset$;
  Compute $\Gamma^{a,o}$ projections of V_{n-1};
  for $b \in \mathcal{B}$ **do**
    $V_n \leftarrow \bigcup backup(b)$;
  $V_{n-1} \leftarrow V_n$;
until $n < N$ ou $|| \max_{\alpha_n \in V_n} \alpha_n \cdot b - \max_{\alpha_{n-1} \in V_{n-1}} \alpha_{n-1} \cdot b || < \epsilon, \forall b \in \mathcal{B}$;

PBVI and the Tiger Problem



Reward function:
- penalty for wrong opening: -100
- reward for correct opening: $+10$
- cost for listening action: -1

Observations:
- to hear the tiger from left door: HL
- to hear the tiger from right door: HR



- small number of reachable beliefs states given the model dynamics
- for this small example the optimal solution is achieved

```
# Tiger problem POMDP description in Cassandra format

discount: 0.95
values: reward
states: tiger-left tiger-right
actions: listen open-left open-right
observations: tiger-left tiger-right

T: listen
identity

T: open-left
uniform

T: open-right
uniform

O: listen
0.85 0.15
0.15 0.85

O: open-left
uniform

O: open-right
uniform

R: listen : tiger-left : * : * -1

R: listen : tiger-right : * : * -1

R: open-left : tiger-left : * : * -100

R: open-left : tiger-right : * : * 10

R: open-right : tiger-left : * : * 10

R: open-right : tiger-right : * : * -100
```

customized PyPOMDP library

- implements PBVI and POMCP algorithms
- see `https://github.com/namoshizun/PyPOMDP`

HSVI : Heuristic Search Value Iteration

(Smith and Simmons, 2004, 2005)

- with **bound-based heuristics** HSVI drives the trajectories on belief state space
  $\rightarrow$ by choosing successively promising actions and observations favoring the error decrease on bounds
- it stops the *in depth* search when the error between bounds is small ($\epsilon\gamma^{-t}$) w.r.t to the value approximation of the initial belief state
- it **updates the value of the upper and lower bounds at each belief state** of the trajectory (from the depth one until the initial - by recursivity)

---

input : POMDP
output: value function V
Initialize $\overline{V}$ et $\underline{V}$;
while $\overline{V}(b_0) - \underline{V}(b_0) > \epsilon$ **do**
  Explore($b_0, \overline{V}, \underline{V}$);

procedure Explore($b, \overline{V}, \underline{V}$);
if $\overline{V}(b) - \underline{V}(b) > \epsilon\gamma^{-t}$ **then**
  $a^* \leftarrow \arg\max_{a \in A} Q_{\overline{V}}(b, a)$;
  $o^* \leftarrow$
    $\arg\max_{o \in \Omega} p(o|b, a^*)(\overline{V}(b_a^o) - \underline{V}(b_a^o))$;
  Explore($b_{a^*}^{o^*}, \overline{V}, \underline{V}$) ;
  $\underline{V} \leftarrow backup(b, \underline{V})$;
  $\overline{V} \leftarrow \mathcal{L}\overline{V}$;

This algorithm inspired the development of others like: FSVI (Shani et al., 2007), SARSOP (Kurniawati et al., 2008) and AEMS (Ross et al., 2008).
Algorithm source code is available: `https://github.com/trey0/zmdp`

POMCP: Partially Observable Monte-Carlo Planning

(Silver and Veness, 2010)

- Online solving algorithm
- Each *trial* starts in b_0.
- Uses UCB1 Auer et al. (2002) for selecting the action to explore

$$\bar{a} \leftarrow \arg\max_{a \in A} \left(Q(h,a) + c\sqrt{\frac{\log(N(h))}{N(ha)}} \right)$$

- Receives the reward and an observation
- *Trial* stops when a leaf state is reached. Initializes the Q-Values for this node with *rollout* method
- Approximates the Q-value which the average accumulated return

$$Q(h,a) = Q(h,a) + \frac{Q(h,a)' - Q(h,a)}{N(ha)}$$

- Stops when planning budget is reached

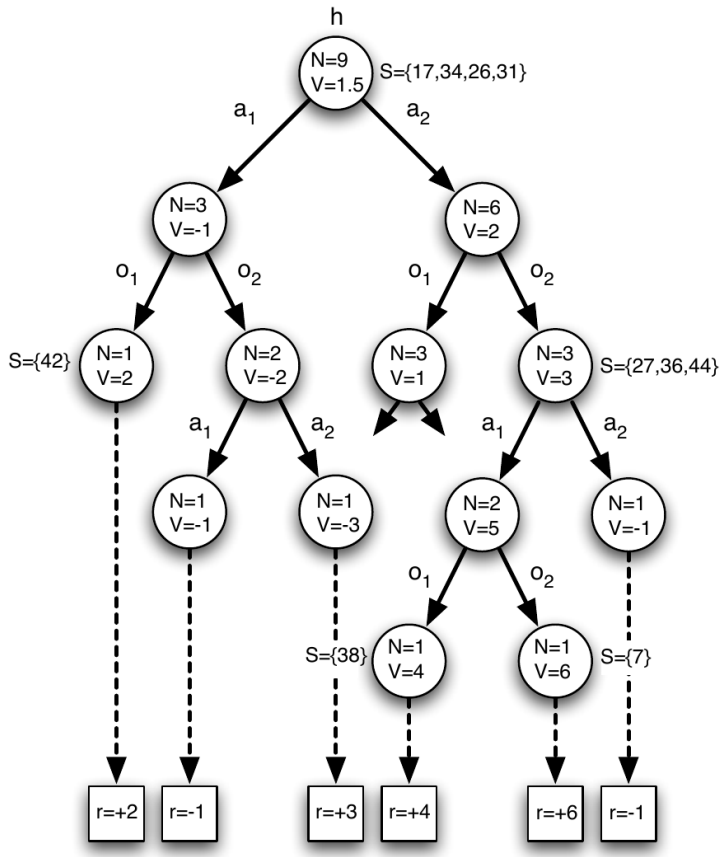


Figure from (Silver and Veness, 2010).

This algorithm inspired recent others like: POMCP-GO (Delamer, 2019), minPOMCP-GO (Carmo et al., 2020)

POMCP: Partially Observable Monte-Carlo Planning

(Silver and Veness, 2010)

- Online solving algorithm
- Each *trial* starts in b_{root}.
- **Uses UCB1 Auer et al. (2002) for selecting the action to explore**
  $\bar{a} \leftarrow \arg\max_{a \in A} \left(Q(h,a) + c\sqrt{\frac{\log(N(h))}{N(ha)}} \right)$
- Receives the reward and an observation
- *Trial* stops when a terminal state is reached.
- Approximates the Q-value which the average accumulated return
  $Q(h,a) = Q(h,a) + \frac{Q(h,a)' - Q(h,a)}{N(ha)}$
- Stops when planning budget is reached

- **The exploration coefficient c forces the algorithm to explore actions that seem less promising.**
  The value of c directly influences the policy convergence.
  However, the constant c is usually problem-dependent.
  This is a strong limitation for real-world further online planning.

This algorithm inspired recent others like: POMCP-GO (Delamer, 2019), minPOMCP-GO (Carmo et al., 2020)

POMCP: Partially Observable Monte-Carlo Planning

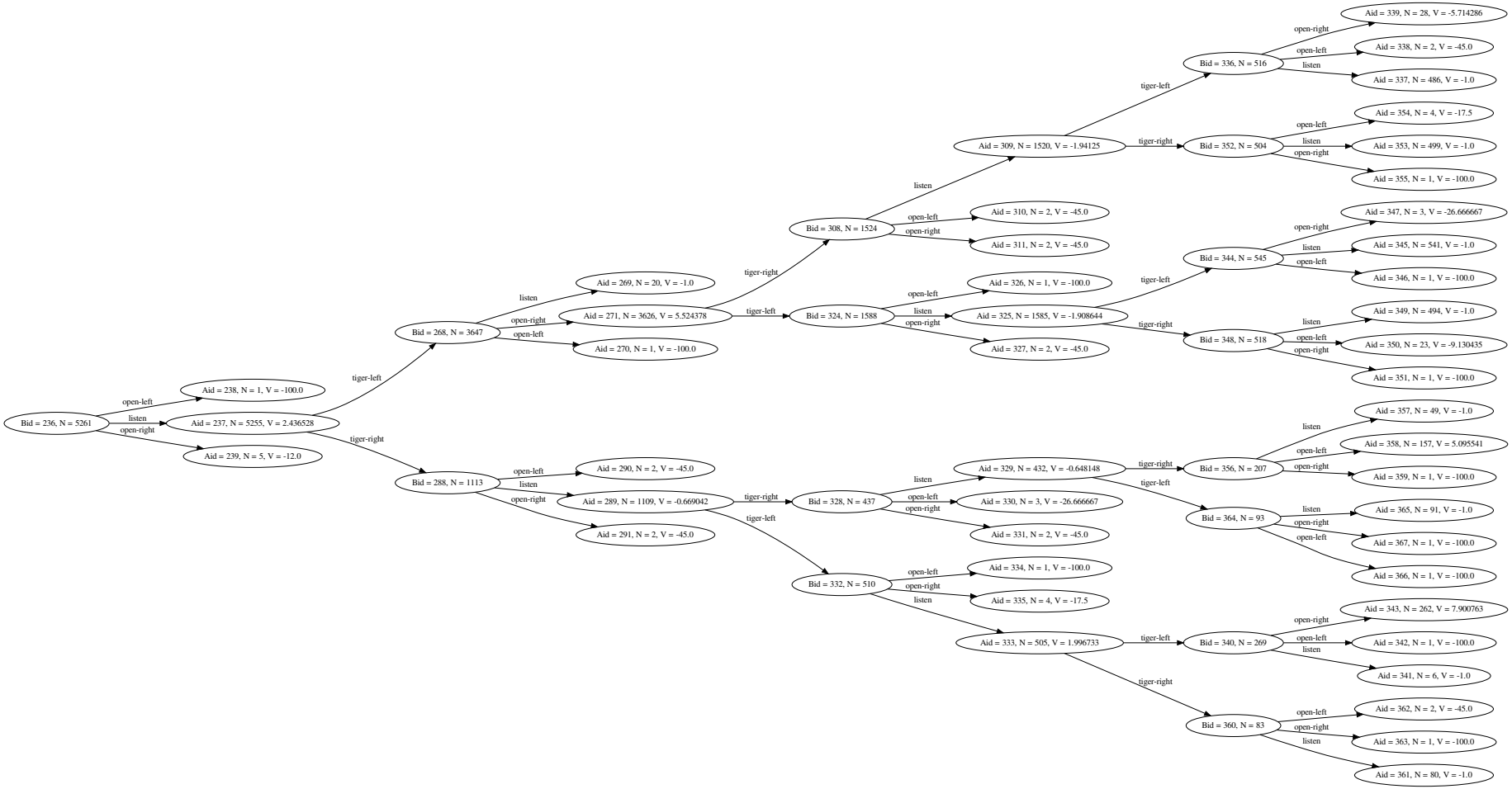
(Silver and Veness, 2010)

- Online solving algorithm
- Each *trial* starts in b_{root}.
- Uses UCB1 ((Auer et al., 2002)) for selecting the action to explore
  $\bar{a} \leftarrow \arg\max_{a \in A} \left(Q(h,a) + c\sqrt{\frac{\log(N(h))}{N(ha)}} \right)$
- Receives the reward and an observation
- *Trial* stops when a terminal state is reached.
- **Approximates the Q-value which the average accumulated return**
  $Q(h,a) = Q(h,a) + \frac{Q(h,a)' - Q(h,a)}{N(ha)}$
- Stops when planning budget is reached

- The exploration coefficient c forces the algorithm to explore actions that seem less promising.
  The value of c directly influences the policy convergence.
  However, the constant c is usually problem-dependent.
  This is a strong limitation for real-world further online planning.
- **The mean return from all trajectories started from the history h when action a is selected, induces a bias in the Q-value approximation (Keller and Helmert, 2013).**
  It also directly influences the value and policy convergence, possibly inducing a non-efficient action choice during path execution.

This algorithm inspired recent others like: POMCP-GO (Delamer, 2019), minPOMCP-GO (Carmo et al., 2020)

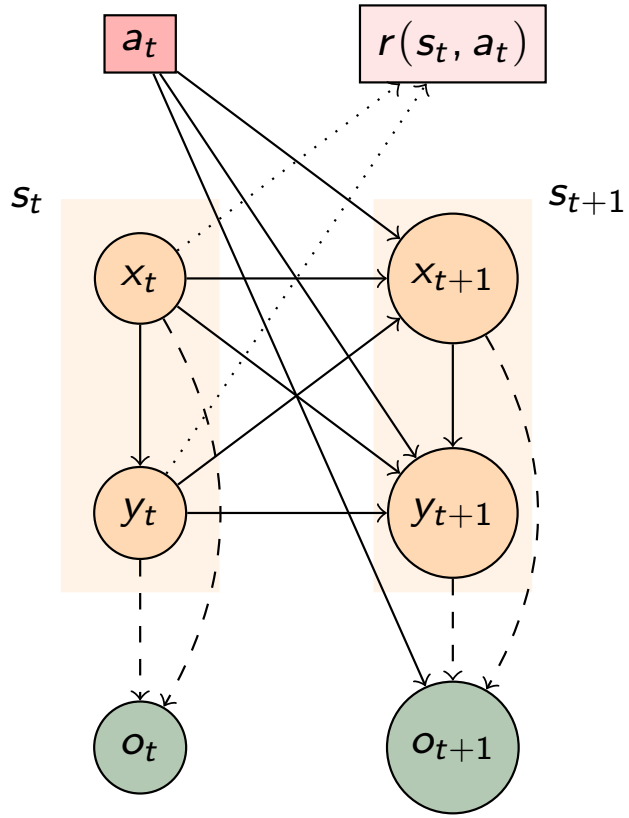
POMCP belief tree for the tiger problem



MOMDP : Mixed Obervability MDP (Ong et al., 2009)

A MOMDP is a tuple $(\mathcal{X}, \mathcal{Y}, A, \Omega, T_{\mathcal{X}}, T_{\mathcal{Y}}, \Omega, R, b_0, \gamma)$, where:

- $\mathcal{X}$ is the bounded set of fully observable state variables;
- $\mathcal{Y}$ is the bounded set of partially observable state variables;
- A is a bounded set of actions;
- Ω is a bounded set of observations;
- $T_{\mathcal{X}} : \mathcal{X} \times A \times \mathcal{X} \times \mathcal{Y} \to [0; 1]$ is a transition function such that $T_{\mathcal{X}}(x, y, a, x') = p(x'|x, y, a)$;
- $T_{\mathcal{Y}} : \mathcal{Y} \times \mathcal{X} \times A \times \mathcal{X} \times \mathcal{Y} \to [0; 1]$ is a transition function such that $T_{\mathcal{Y}}(x, y, a, x', y') = p(y'|x, y, a, x')$;
- $O : \Omega \times \mathcal{Y} \to [0; 1]$ is an observation function such that $O(o, a, x', y') = p(o|x', y', a)$;
- $R : \mathcal{X} \times \mathcal{Y} \times A \to \mathbb{R}$ is a reward function associated with a state-action pair; and:
- $b_0 = (x_0, b_{\mathcal{Y}_0})$ is the initial probability distribution over states.
- $\gamma \in [0, 1[$ is the discount factor.



MOMDP : Mixed Obervability MDP (Ong et al., 2009)

Belief state update redefinition

$$b_{\mathcal{Y}}^{o,a,x'}(y') = \eta \sum_{y' \in \mathcal{Y}} p(o|y', x', a) p(y'|x, y, a, x') p(x'|x, y, a) b_{\mathcal{Y}}(y)$$

where, η is a normalization constant. The belief state b is now noted by the couple $(x, b_{\mathcal{Y}})$, and $\mathcal{B}_{\mathcal{Y}}$ is the belief state space y conditioned by x : $\mathcal{B}_{\mathcal{Y}}(x) = \{(x, b_{\mathcal{Y}}), b_{\mathcal{Y}} \in \mathcal{B}_{\mathcal{Y}}\}$. $\mathcal{B}_{\mathcal{Y}}(x)$ is a sub-space of $\mathcal{B}$, such that $\mathcal{B} = \bigcup_{x \in \mathcal{X}} \mathcal{B}_{\mathcal{Y}}(x)$.

Solving MOMDPs consists in finding *a set of policies* $\pi_x : \mathcal{B}_{\mathcal{Y}} \rightarrow A$

$$\pi_x^* \leftarrow \underset{\pi_x \in \Pi}{\arg\max}\, E_{\pi_x} \left[\sum_{t=0}^{\infty} \gamma^t r((x_t, b_{\mathcal{Y}}^t), \pi((x_t, b_{\mathcal{Y}}^t))) \middle| b_0 = (x_0, b_{\mathcal{Y}_0}) \right]$$

As for the POMDP, the value function at a time step $n < \infty$ can be also represented by a set of α-vectors:

$$V_n(x, b_{\mathcal{Y}}) = \max_{\alpha \in \Gamma_{\mathcal{Y}}^n(x)} (\alpha \cdot b_{\mathcal{Y}})$$

where α is the hyperplan over the space $\mathcal{B}_y(x)$. In this way, the value function over the complete state space is parametrized by the set $\Gamma_{\mathcal{Y}}(x)$, i.e. $\Gamma = \{\Gamma_{\mathcal{Y}}(x), x \in \mathcal{X}\}$.

MOMDP : Mixed Obervability MDP (Ong et al., 2009)

Belief state update redefinition

$$b_{\mathcal{Y}}^{o,a,x'}(y') = \eta \sum_{y' \in \mathcal{Y}} p(o|y', x', a) p(y'|x, y, a, x') p(x'|x, y, a) b_{\mathcal{Y}}(y)$$

where, η is a normalization constant. The belief state b is now noted by the couple $(x, b_{\mathcal{Y}})$, and $\mathcal{B}_{\mathcal{Y}}$ is the belief state space y conditioned by x : $\mathcal{B}_{\mathcal{Y}}(x) = \{(x, b_{\mathcal{Y}}), b_{\mathcal{Y}} \in \mathcal{B}_{\mathcal{Y}}\}$. $\mathcal{B}_{\mathcal{Y}}(x)$ is a sub-space of $\mathcal{B}$, such that $\mathcal{B} = \bigcup_{x \in \mathcal{X}} \mathcal{B}_{\mathcal{Y}}(x)$.

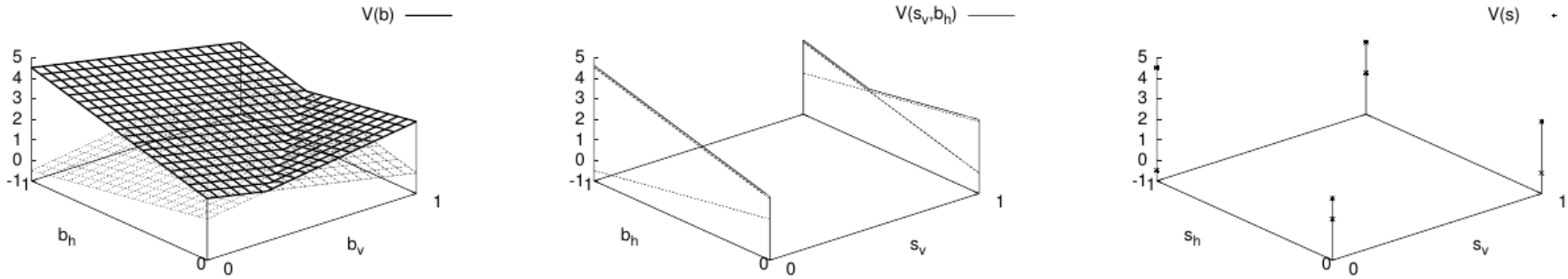


Figure from (Araya-López et al., 2010). In order: a POMDP value function representation, a MOMDP value function representation and the MDP value representation (dots).

For more details about MOMDP efficient solving algorithm, see Ong et al. (2009, 2010).

Outline

Human-robot(s) systems

Increasing use of automated systems

assembly lines, autopilots in aircrafts, autonomous cars, unmanned vehicles: drones/ground robots for military operation/contaminated area ...

- **Increasingly decisional autonomy:**
  technical advances in AI, statistical learning, vision, decision-making ...
- **Human operator still vital:**
  - produces tactical, moral, social and ethical decisions
  - flexible/creative, handles complex/unknown situations
  - complementary strengths (Fitts list (de Winter and Dodou, 2014))
  - need for people for responsibility assessment issues

Mixed-Initiative framework

defines the role of the human and artificial agents according to their recognized skills (Adams et al., 2004; Allen et al., 1999).

Human operator weaknesses

Human operator weaknesses

For instance: the percentage of involvement of human factors in UAVs operations varied across aircraft from 21% to 68%. (U.S. Army, Navy, and Air Force report (Williams, 2004))

Constraints experienced by humans

- pressure (e.g. cause by a danger) $\rightarrow$ stress
- task complexity $\rightarrow$ cognitive load
- task hardness and duration $\rightarrow$ fatigue, boredom

Consequences:

- disengagement, lower vigilance
- mind wandering
- over-engagement, attentional tunneling
- mental confusion... **affect human abilities!**

Increase in accident risk resulting in mission fails or sub-optimal achievements

To improve mixed-initiative missions

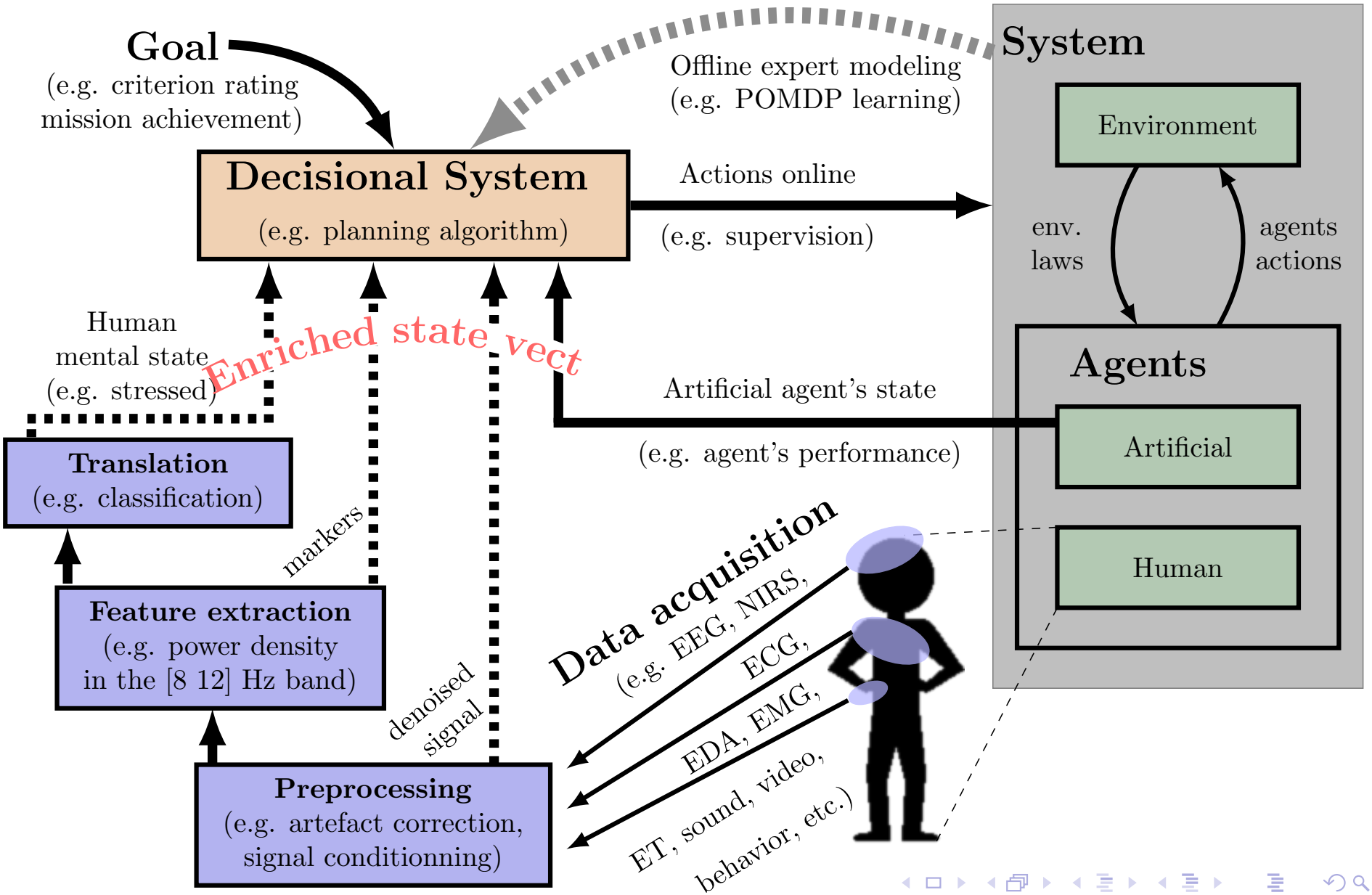
Human operators are not providential agents

- not always reliable
- not always omniscient during the mission
- not always able to fix any occurring issue

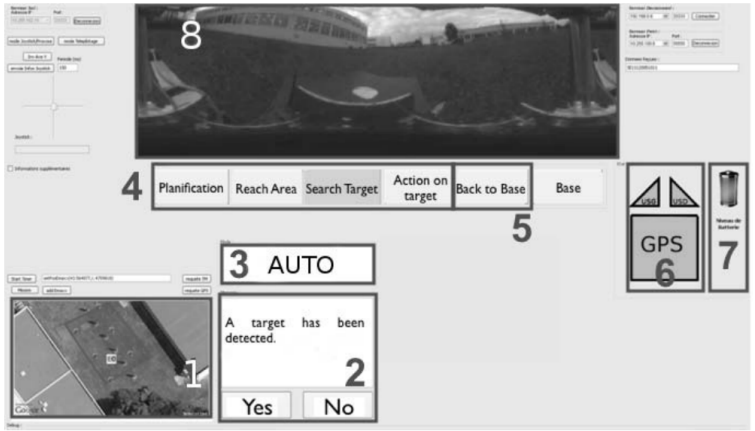
A possible solution: Mixed-initiative strategy computation

it should define the role of the human and artificial agents according to their recognized skills and current capabilities (Chanel et al., 2020b).
$\rightarrow$ need for **monitoring the operator**'s cognitive state (mental resource)
$\rightarrow$ then **decide** when the automated system should or should not have the authority
$\rightarrow$ **adaptation** of machine's behavior

Cybernetical System to drive human and artificial agent(s) interactions



Mission planning considering the human operator's cognitive state

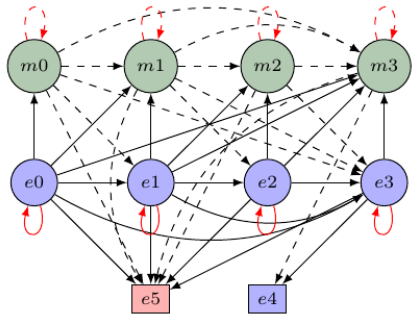


- **A MOMDP mission model considering human and robot agents (cap-)abilities (de Souza et al., 2015)**
  → robot sensors failures:
  → behavioral statistics (e.g. ET) and observed cognitive availability (e.g. ECG + ET) of the human operator
- **Goal:** to proceed the mission planning in order to maximize mission achievements.
  Visible state variables: Mission phases × robot operation mode × sensors status
  PO state variable: Operator's cognitive availability
  Actions: `goToZone`, `tgtSearch`, `tgtHandle`, `retBase`, `onBase`, `getAttention`, `cMeasure`
  Observation: `oAvailable`

SENSORS STATUSES

Sensor	Ok	Not Ok	FP
Battery	battOK	battKO	2/30
GPS[1]	gpsOK	gpsKO	3/30
Ultrasound[1]	usOK	usKO	3/30
Camera[2]	camOK	camKO	3/30
Ground station[2]	stOK	stKO	1/30
Joystick[2]	jsOK	jsKO	1/30
Xbee[2]	xbOK	xbKO	2/30

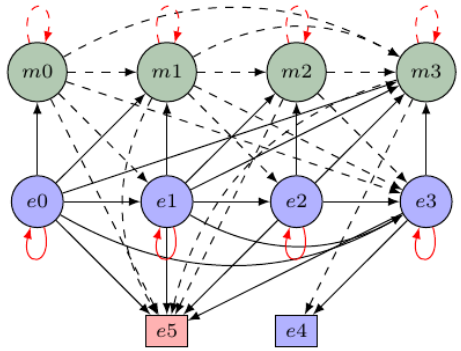
[1] indicates the essential sensors for autonomous operation.
[2] indicates the essential devices for manual operation.



	available_N	available_Y
oAvailable_N	91.1	8.9
oAvailable_Y	0	100

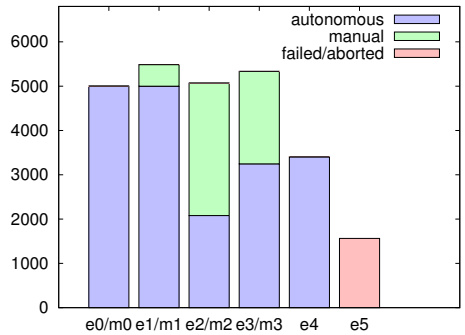
Mission planning considering the human operator's cognitive state

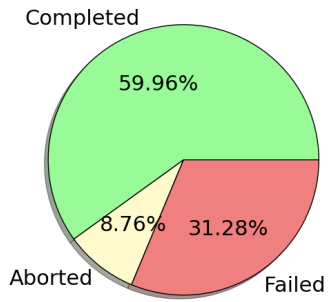
mission phase	autonomous mode	manual mode
going to zone	e0	m0
target searching	e1	m1
target handling	e2	m2
returning to base	e3	m3
on base (final)	e4	-
failed (final)	e5	-



Approach advantages and drawbacks

- model (partially) learned from collected data acquired in previous experiments
- cognitive availability observation relied on a binary classifier output also trained with collected data
- observation function is then the confusion matrix of such a binary classifier
- the approach was not evaluated *in situ*, only in simulation (missing real-time monitoring system)



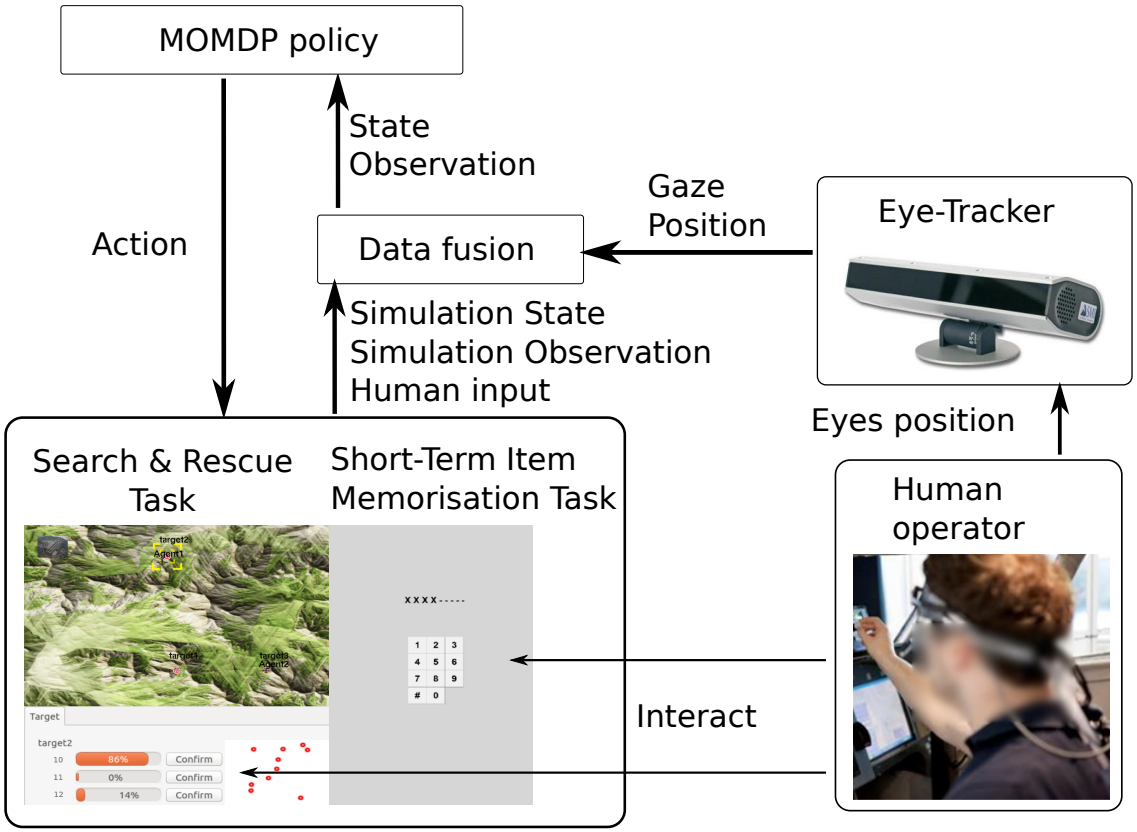


Considering the human's availability and non-deterministic behavior

(Gateau et al., 2016)

Online evaluation and estimation of operator's state

→ UAVs adaptive behavior in function of the availability of the operator
→ Availability estimated with eye-tracker
→ 10 participants, 4 experimental conditions (10min)

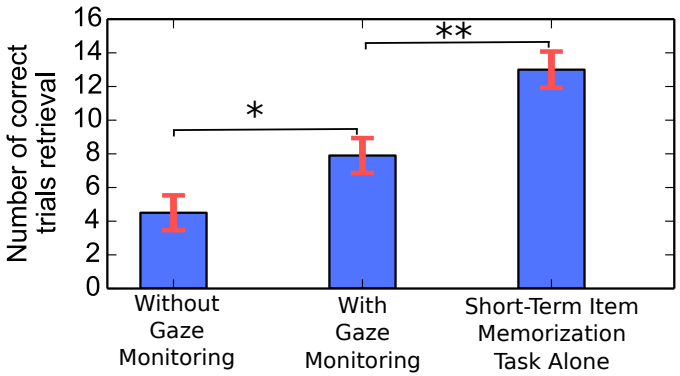


Considering the human's availability and non-deterministic behavior

(Gateau et al., 2016)

- **Results**

Performance Memorization task



Average scores Search & Rescue task (and STD).

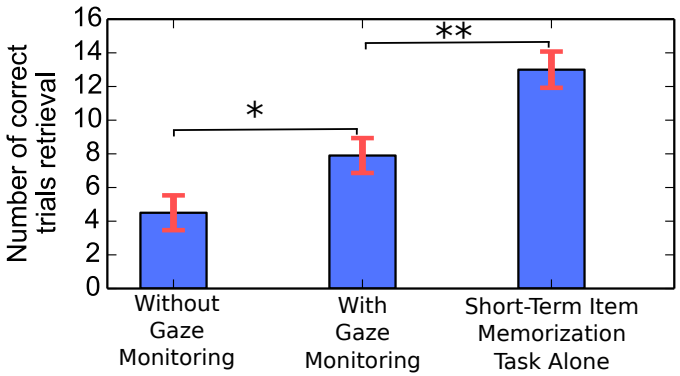
Metric	Request ratio score	Averaged discounted sum of rewards
without GM	0.76 (0.26)	1321.4 (456.5)
with GM	0.84 (0.11)	1210.4 (269.5)
S&R task alone	0.92 (0.03)	1143.4 (469.9)

Considering the human's availability and non-deterministic behavior

(Gateau et al., 2016)

- **Results**

  Performance Memorization task



  Average scores Search & Rescue task (and STD).

Metric	Request ratio score	Averaged discounted sum of rewards
without GM	0.76 (0.26)	1321.4 (456.5)
with GM	0.84 (0.11)	1210.4 (269.5)
S&R task alone	0.92 (0.03)	1143.4 (469.9)

- **Limitations**

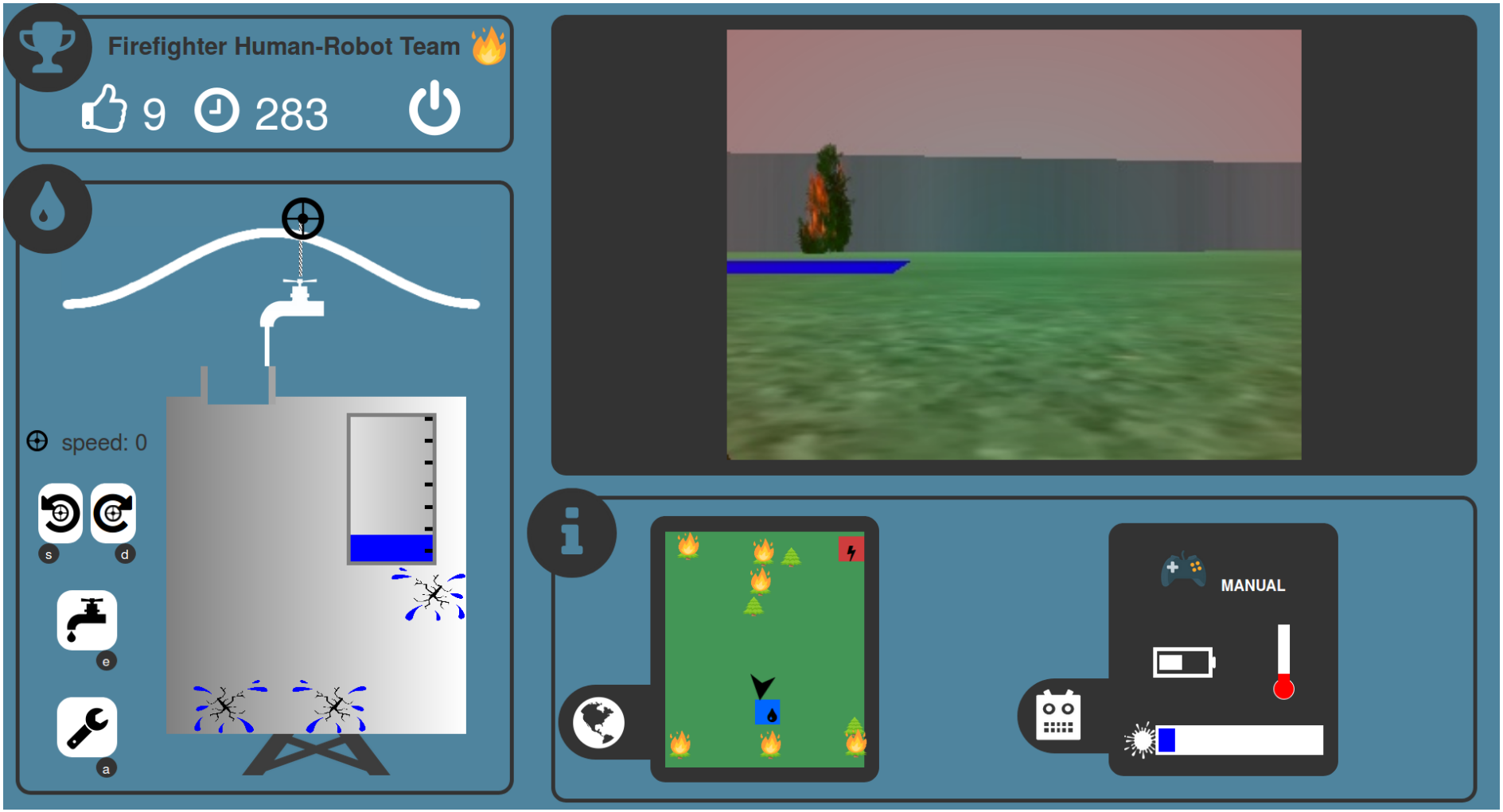
  → Based on **"fake" interaction data** (e.g operator time to answer, identification correctness)

  → A dual-task paradigm with **independent performance metrics** - it had not considered a mission common goal

  → Availability was estimated with ET what gave **where the human was looking** at time step t, **it does not inform about the real human's (un)availability or cognitive load** which is a **a partial observable** state by definition.

The Firefighter Robot Game

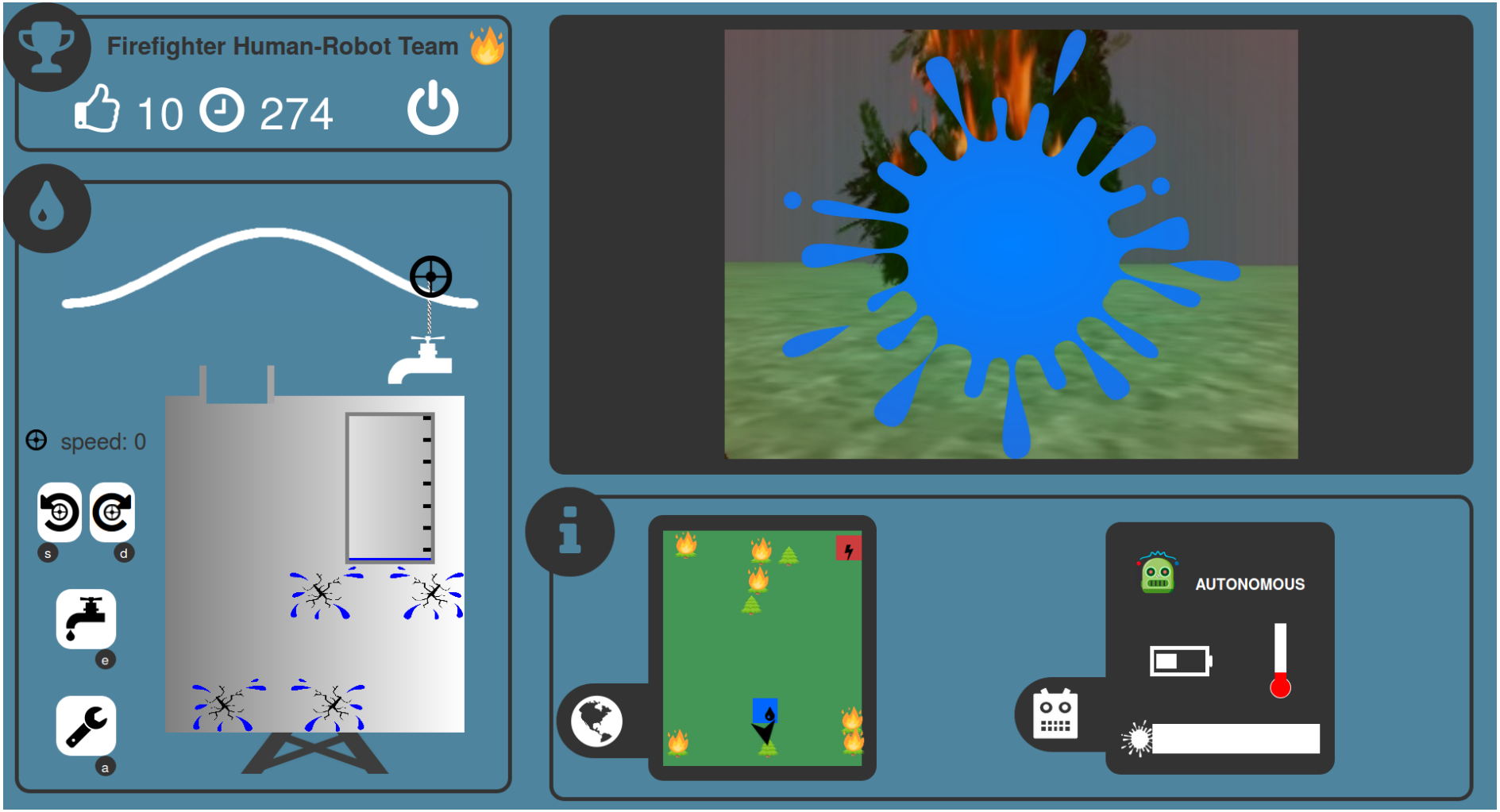
Graphical user interface (GUI)



have a look at: **http://robot-isae.isae.fr/**

The Firefighter Robot Game

Graphical user interface (GUI)



have a look at: **http://robot-isae.isae.fr/**

The Firefighter Robot Game

Displayed alarms

have a look at: **http://robot-isae.isae.fr/**

- A serious game to obtain a **balanced dataset** (with & without autonomy/alarms)
  - $\rightarrow$ random robot's operating mode (auto/manual control)
  - $\rightarrow$ random display of information to the operator

- pre-testing for:
  - $\rightarrow$ **degraded human behaviors** emergence:
    - water management + robot's control $\rightarrow$ demanding
    - score/limited time to $\rightarrow$ pressure
    - battery empty or too hot robot $\rightarrow$ danger

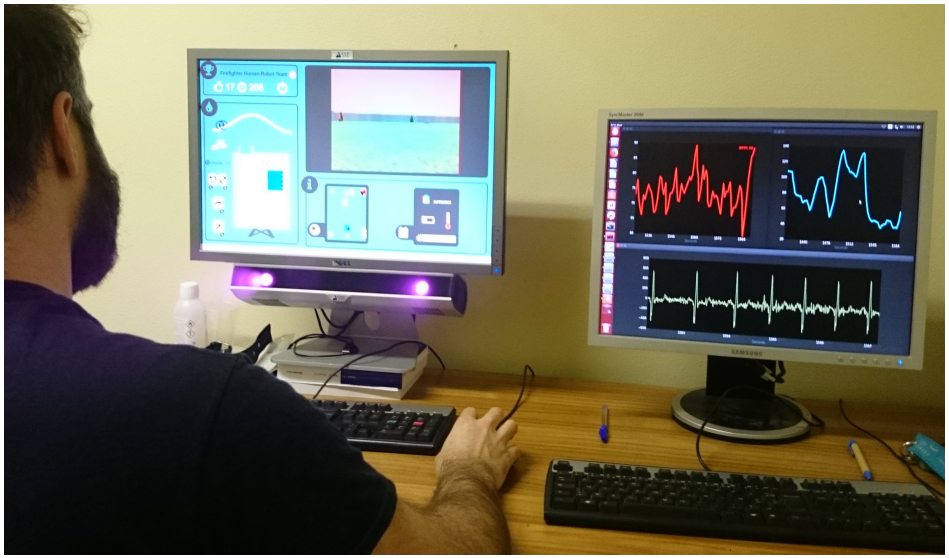
A dataset on Human-Agent Interaction

data on human-machine interaction

- total recorded time: 86h 25m 54s, with human actions: 48h 13m 26s.
- available at: `https://personnel.isae-supaero.fr/isae_ressources/caroline-chanel/horizon/`
- 18 participants in lab (equipped with physiological sensors) - not available

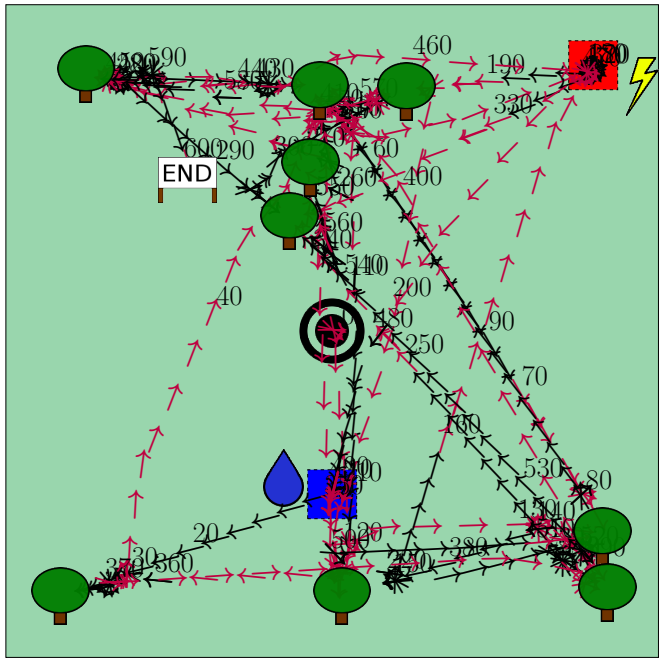
Could be useful for:

- human-robot team supervision in a mixed-initiative framework;
- learning with demonstration;
- study of human factors, human behavior, physiological/behavioral data analysis;

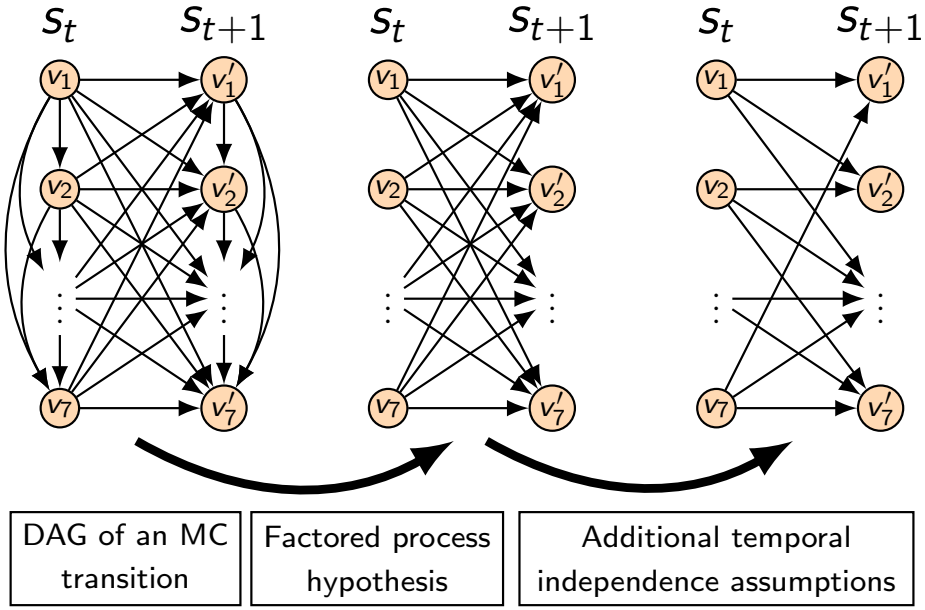


MDP Interaction model learning based on crowdsourcing

(Charles et al., 2018)



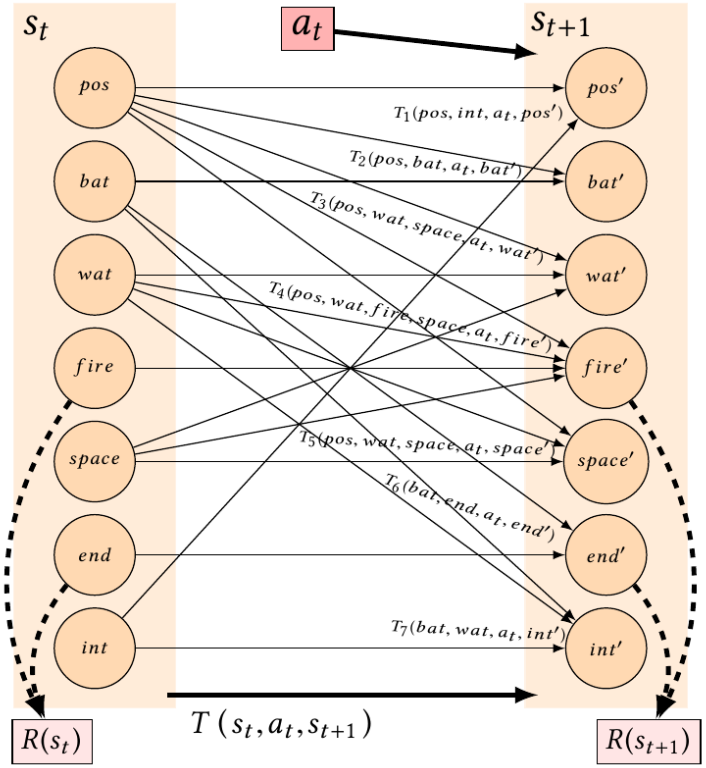
Example of game trial



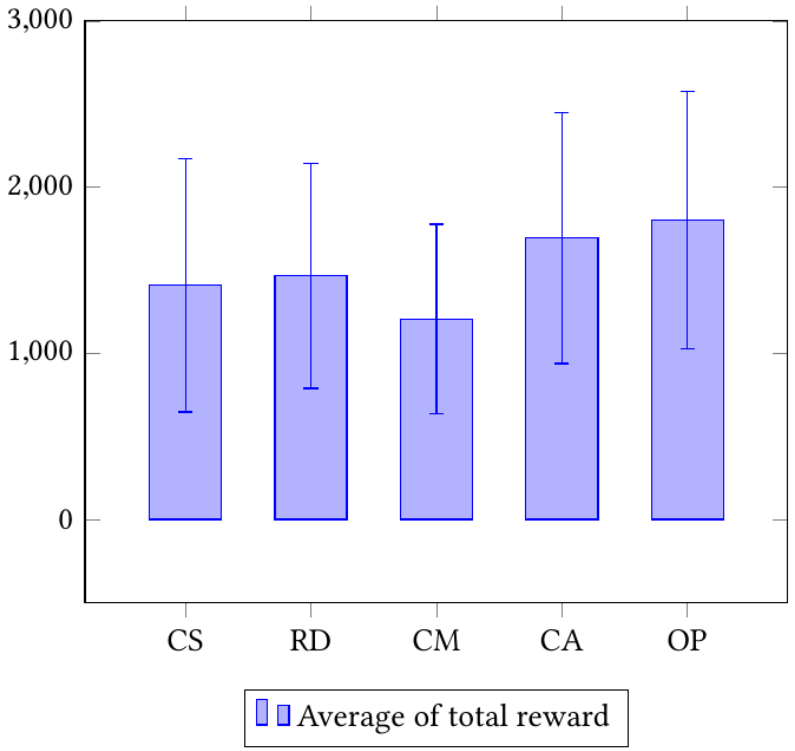
Transition, factorization and in dependency assumptions

MDP Interaction model learning based on crowdsourcing

(Charles et al., 2018)

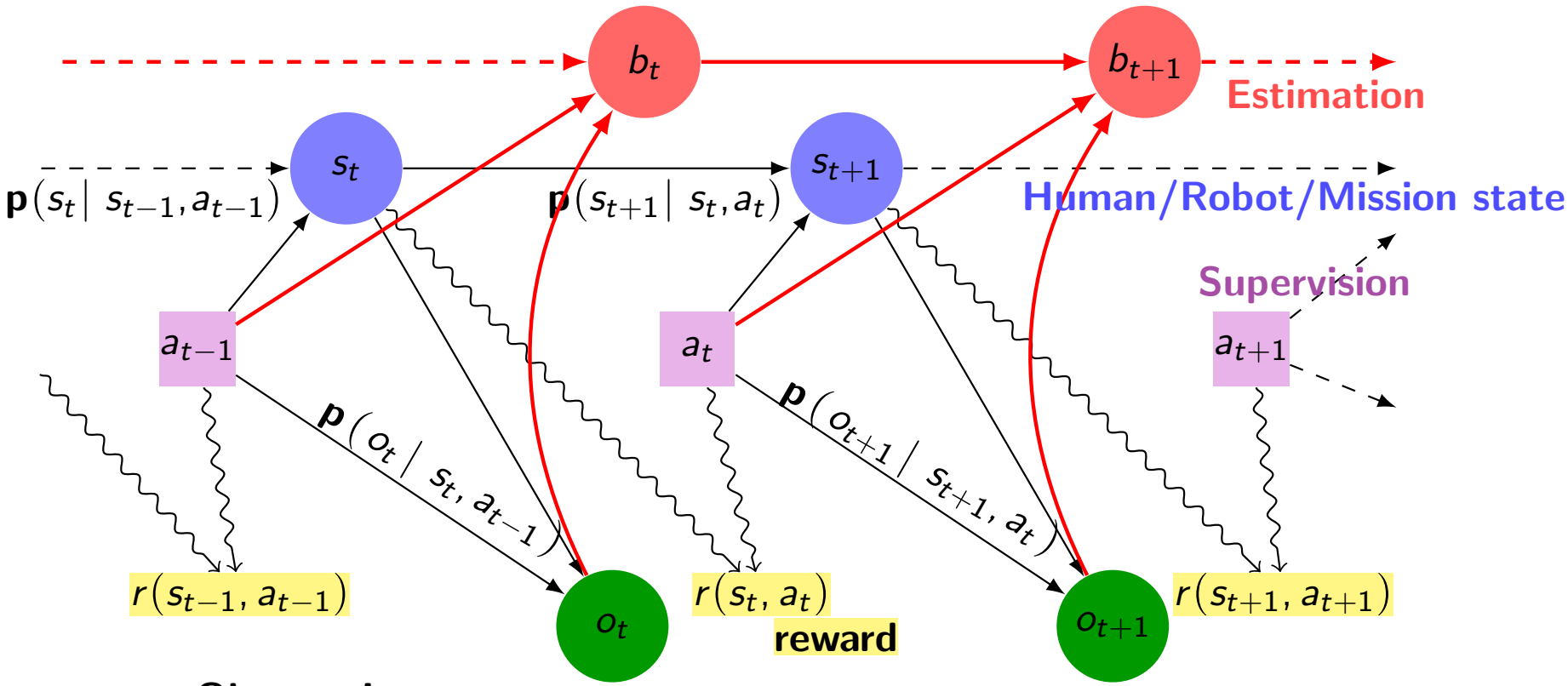


Final factored MDP model.
Transition probabilities learnt with PomeGranate library.



Policy evaluation in simulation.
Policy computed with PROST (Keller and Eyerich, 2012)

POMDP for driving HMI using human-related observations

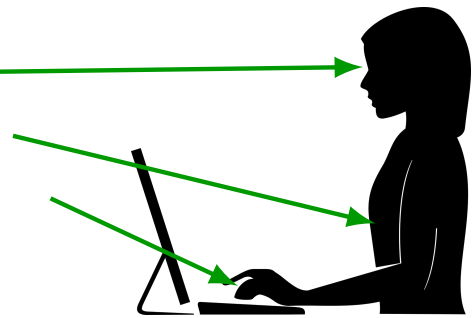


Observations o_t:

Eye tracking (ET)

Electrocardiography (ECG)

Human actions on the interface (HAI)



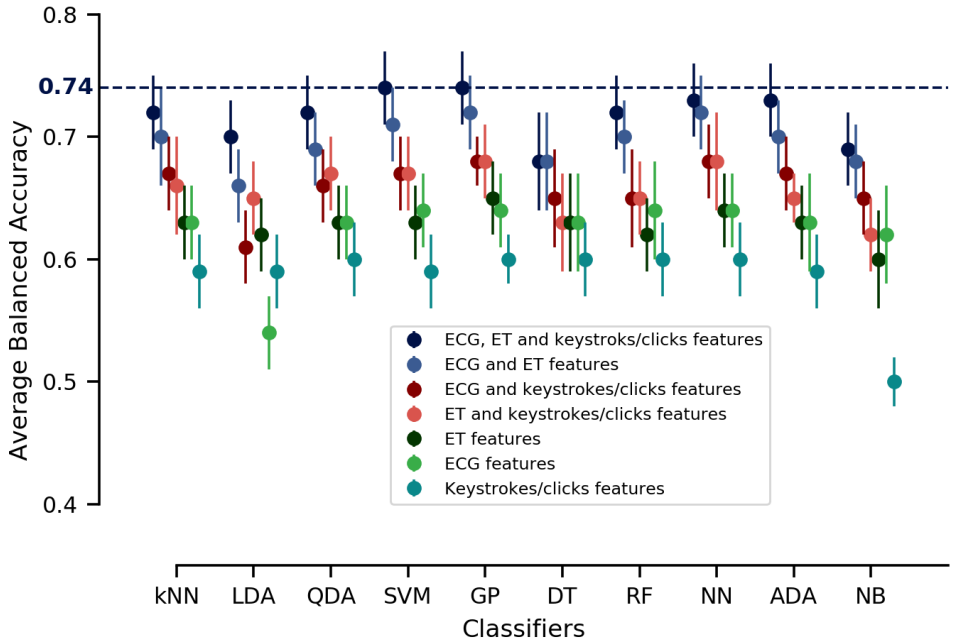
Would the physiological/behavioral data be useful?

A recent work (Chanel et al., 2020a) showed:

High-scoring *vs* low-scoring missions

→ significant differences on behavioral and physiological (ECG, ET) markers were observed
→ robot autonomous mode in high-scoring missions was detrimental to mission performance, while helpful in low-scoring missions

It allowed to train classifiers in order to approach the observation function $p(o|s)$ informing about the joint-performance given the current ET, ECG and HAI markers and robot operation mode.



Let's try it !

Practical work proposition (12h)

- Propose a (simple yet effective) POMDP model to drive the interaction for the Firefighter Robot Game study case.
- Using the data collected with the Firefighter Robot Game:
  - define state, action, and observation sets.
  - learn the transition and the observation functions.
  - proceed with classifiers training to **observe** the human operator engagement based on ET, ECG and HAI data.
  - suggestion : use the resulting confusion matrix as observation function
- The resulting policy should decide at each time step (about 10s): the robot operation mode, as well as, when to launch alarms.
- Evaluate the model in simulation

Bibliographie I

Adams, J. A., Rani, P., and Sarkar, N. (2004). Mixed initiative interaction and robotic systems. In *AAAI Workshop on Supervisory Control of Learning and Adaptive Systems*, page 613.

Albore, A., Ramirez, M., and Geffner, H. (2011). Effective heuristics and belief tracking for planning with incomplete information. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Freiburg, Germany.

Allen, J. E., Guinn, C. I., and Horvtz, E. (1999). Mixed-initiative interaction. *IEEE Transactions on Intelligent Systems and their Applications*, 14(5):14–23.

Araya-López, M., Thomas, V., Buffet, O., and Charpillet, F. (2010). A closer look at momdps. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 197–204. IEEE.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Barto, A., Bradtke, S., and Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138.

Bellman, R. (1954). The theory of dynamic programming. Technical report, RAND Corp Santa Monica CA.

Bertoli, P., Cimatti, A., Roveri, M., and Traverso, P. (2006). Strong planning under partial observability. *Artificial Intelligence*, 170(4-5):337–384.

Bertsekas, D. and Castanon, D. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108.

Bibliographie II

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1.

Blum, A. L. and Furst, M. L. (1995). Fast planning through planning graph analysis. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.

Bonet, B. (2003). Labeled rtdp: Improving the convergence of real-time dynamic programming. In *Proc. ICAPS*.

Bonet, B. and Geffner, H. (2009). Solving POMDPs: RTDP-bel vs. point-based algorithms. In *International Joint Conference on Artificial Intelligence (IJCAI)*, San Francisco, CA, USA.

Brafman, R. and Hoffmann, J. (2004). Conformant planning via heuristic forward search: A new approach. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Whistler, Canada.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Bryce, D., Kambhampati, S., and Smith, D. E. (2006). Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research (JAIR)*, 26:35–99.

Carmo, A. R., Delamer, J.-A., Watanabe, Y., Ventura, R., and Ponzoni Carvalho Chanel, C. (2020). Entropy-based adaptive exploit-explore coefficient for monte-carlo path planning. In *10th International Conference on Prestigious Applications of Intelligent Systems, a subconference of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, pages 1–8.

Bibliographie III

Cassandra, A. (1998). *Exact and approximate algorithms for partially observable Markov decision processes*. PhD thesis, Brown University Providence, RI, USA.

Chanel, C. P., Roy, R. N., Dehais, F., and Drougard, N. (2020a). Towards mixed-initiative human–robot interaction: Assessment of discriminative physiological and behavioral features for performance prediction. *Sensors*, 20(1):296.

Chanel, C. P., Roy, R. N., Drougard, N., and Dehais, F. (2020b). Mixed-initiative human-automated agents teaming: Towards a flexible cooperation framework. In *International Conference on Human-Computer Interaction*, pages 117–133. Springer.

Charles, J.-A., Chanel, C. P., Chauffaut, C., Chauvin, P., and Drougard, N. (2018). Human-agent interaction model learning based on crowdsourcing. In *Proceedings of the 6th International Conference on Human-Agent Interaction*, pages 20–28.

de Souza, P. E. U., Chanel, C. P. C., and Dehais, F. (2015). Momdp-based target search mission taking into account the human operator's cognitive state. In *2015 IEEE 27th international conference on tools with artificial intelligence (ICTAI)*, pages 729–736. IEEE.

de Winter, J. C. F. and Dodou, D. (2014). Why the fitts list has persisted throughout the history of function allocation. *Cognition, Technology & Work*, 16(1):1–11.

Delamer, J.-A. (2019). *Planification de stratégies de navigation et de guidage pour des drones autonomes dans des milieux encombrés*. PhD thesis, Institut Supérieur de l'Aéronautique et de l'Espace (ISAE).

Bibliographie IV

Frances, G., Geffner, H., Lipovetzky, N., and Ramirez, M. (2018). Best-first width search in the ipc 2018: Complete, simulated, and polynomial variants. *IPC2018–Classical Tracks*, pages 22–26.

Gateau, T., Chanel, C. P. C., Le, M.-H., and Dehais, F. (2016). Considering human's non-deterministic behavior and his availability state when designing a collaborative human-robots system. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4391–4397. IEEE.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D., Barrett, A., Christianson, D., et al. (1998). Pddl— the planning domain definition language. *Technical Repport Yale Center*.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

Hansen, E. A. and Zilberstein, S. (2001). Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1):35–62.

Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Keidar, M. and Kaminka, G. A. (2012). Robot exploration with fast frontier detection: theory and experiments. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 113–120. International Foundation for Autonomous Agents and Multiagent Systems.

Bibliographie V

Keller, T. and Eyerich, P. (2012). Prost: Probabilistic planning based on uct. In *ICAPS*.

Keller, T. and Helmert, M. (2013). Trial-based heuristic tree search for finite horizon mdps.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Kolobov, A., Dai, P., Mausam, M., and Weld, D. S. (2012). Reverse iterative deepening for finite-horizon mdps with large branching factors. In *Twenty-Second International Conference on Automated Planning and Scheduling*.

Kolobov, A., Weld, D. S., et al. (2016). Lrtdp vs. uct for online probabilistic planning.

Kurniawati, H., Hsu, D., and Lee, W. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. RSS*.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Littman, M. (1997). A. Cassandra and N. Zhang. Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*.

Littman, M., Cassandra, A., and Pack Kaelbling, L. (1995). Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, pages 362–370.

Mausan and Kolobov, A. (2012). Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210.

Bibliographie VI

McAllester, D. and Singh, S. (1999). Approximate planning for factored pomdps using belief state simplification. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 409–416. Morgan Kaufmann Publishers Inc.

Monahan, G. (1982). A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, pages 1–16.

Ong, S. C., Png, S. W., David, and Lee, H. W. S. (2009). POMDPs for robotic tasks with mixed observability. In *Proceedings of Robotics: Science and Systems (RSS)*.

Ong, S. C. W., Png, S. W., Hsu, D., and Lee, W. S. (2010). Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068.

Papadimitriou, C. and Tsitsiklis, J. (1987). The complexity of Markov decision processes. *Mathematics of OR*, 12(3):441–450.

Paquet, S., Chaib-draa, B., and Ross, S. (2006). Hybrid POMDP algorithms. In *Proceedings of The Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*, pages 133–147.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Richter, S. and Westphal, M. (2010). The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.

Bibliographie VII

Rooker, M. N. and Birk, A. (2007). Multi-robot exploration under the constraints of wireless networking. *Control Engineering Practice*, 15(4):435–445.

Ross, S., Pineau, J., Paquet, S., and Chaib-Draa, B. (2008). Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 32(1):663–704.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Pearson Education Limited.

Satia, J. and Lave, R. (1973). Markovian decision processes with probabilistic observation of states. *Management Science*, 20(1):1–13.

Shani, G., Brafman, R., and Shimony, S. (2007). Forward search value iteration for pomdps. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*.

Sigaud, O. and Buffet, O. (2008). Processus décisionnels de markov en intelligence artificielle.

Silver, D. and Veness, J. (2010). Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172.

Smith, T. and Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, Banff, Canada.

Smith, T. and Simmons, R. (2005). Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. UAI*.

Somani, A., Ye, N., Hsu, D., and Lee, W. S. (2013). Despot: Online pomdp planning with regularization. In *Advances in neural information processing systems*, pages 1772–1780.

Bibliographie VIII

Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the Infinite Horizon: Discounted Costs. *Operations Research*, 26(2):282–304.

Spaan, M. and Vlassis, N. (2004). A point-based POMDP algorithm for robot planning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Teichteil-Königsbuch, F., Kuter, U., and Infantes, G. (2010). Incremental plan aggregation for generating policies in mdps. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1231–1238.

Tesauro, G. and Galperin, G. R. (1997). On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074.

Washington, R. (1997). Bi-pomdp: Bounded, incremental partially-observable markov-model planning. *Recent Advances in AI Planning*, pages 440–451.

Watanabe, Y., Veillard, A., and Chanel, C. (2016). Navigation and guidance strategy planning for uav urban operation. In *AIAA Infotech@ Aerospace*, page 0253.

Williams, K. W. (2004). A Summary of Unmanned Aircraft Accident / Incident Data: Human Factors Implications. Technical Report December 2004, DTIC Document.

Yoon, S. W., Fern, A., and Givan, R. (2007). Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, pages 352–359.