INFINITY '09

An Inverse Method for Markov Decision Processes

Étienne André

Laurent Fribourg

Laboratoire Spécification et Vérification

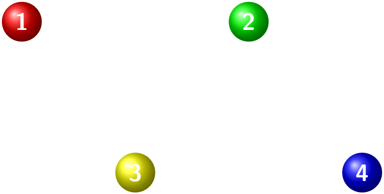
LSV, ENS de Cachan & CNRS

Context: Hardware Verification

- Verification of real time systems with stochastic behavior
  - Need to express probabilities
  - Need to express infinite behaviors
  - Use of Markov decision processes [Bel57, How60]
- Need for adjusting some timings or costs of the system
  - Use of parameters (unknown constants)
  - Definition of a zone of good behavior for the parameters

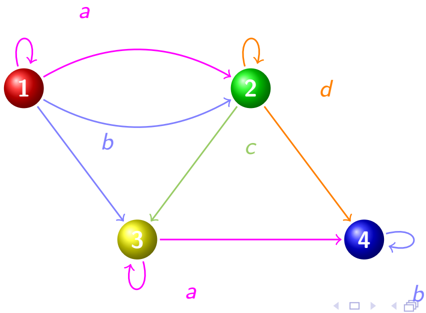
Markov Decision Process (MDP)

- Weighted labeled directed graph augmented with probabilities
  - A set of states $S = \{s_1, \ldots, s_n\}$



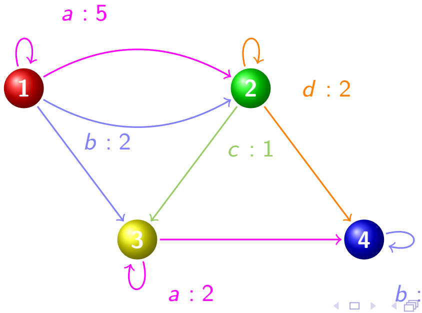
Markov Decision Process (MDP)

- Weighted labeled directed graph augmented with probabilities
  - A set of states $S = \{s_1, \ldots, s_n\}$
  - A set A of actions (or labels)



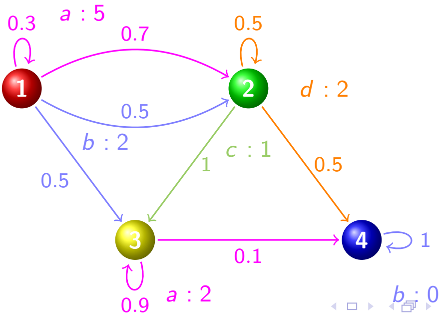
Markov Decision Process (MDP)

- Weighted labeled directed graph augmented with probabilities
  - A set of states $S = \{s_1, \ldots, s_n\}$
  - A set A of actions (or labels)
  - A weight function w, associating a cost $w(s, a)$ to every state s and action a



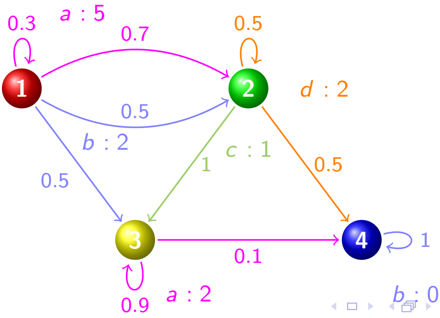
Markov Decision Process (MDP)

- Weighted labeled directed graph augmented with probabilities
  - A set of states $S = \{s_1, \ldots, s_n\}$
  - A set A of actions (or labels)
  - A weight function w, associating a cost $w(s, a)$ to every state s and action a
  - A probability function $Prob$, associating a probability to every edge, such that the sum of the probabilities of leaving a state s through action a is equal to 1, i.e., $\sum_{s' \in S} Prob(s, a, s') = 1$



Markov Decision Process (MDP)

- Weighted labeled directed graph augmented with probabilities
  - A set of states $S = \{s_1, \ldots, s_n\}$, including one absorbing state s_n
  - A set A of actions (or labels)
  - A weight function w, associating a cost $w(s, a)$ to every state s and action a
  - A probability function $Prob$, associating a probability to every edge, such that the sum of the probabilities of leaving a state s through action a is equal to 1, i.e., $\sum_{s' \in S} Prob(s, a, s') = 1$

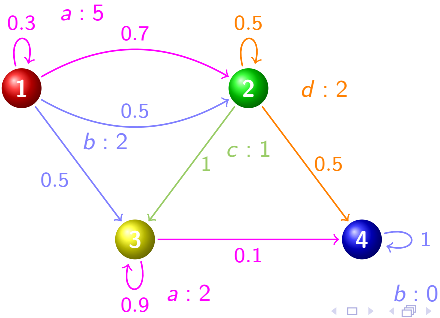


The Direct Problem: Optimal Policy

- Policy μ: function from states to actions $S \rightarrow A$
  - Resolves the non-determinism
  - The MDP becomes a Markov Chain [KMST59]
- Optimal policy: policy such that the sum of the weights until the absorbing state is minimal

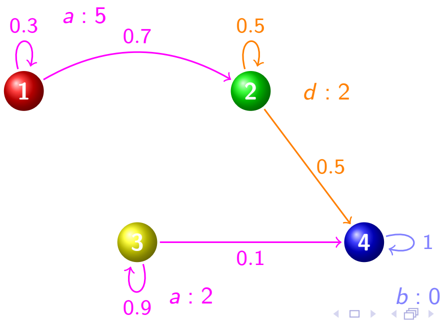
The Direct Problem: Optimal Policy

- Policy μ: function from states to actions $S \rightarrow A$
  - Resolves the non-determinism
  - The MDP becomes a Markov Chain [KMST59]
- Optimal policy: policy such that the sum of the weights until the absorbing state is minimal
- Optimal policy for our example of MDP



The Direct Problem: Optimal Policy

- Policy μ: function from states to actions $S \rightarrow A$
  - Resolves the non-determinism
  - The MDP becomes a Markov Chain [KMST59]
- Optimal policy: policy such that the sum of the weights until the absorbing state is minimal
- Optimal policy for our example of MDP
  - $\mu = \{1 \rightarrow a, \quad 2 \rightarrow d, \quad 3 \rightarrow a\}$



The Inverse Problem

- The direct problem
  - Given an MDP, compute an optimal policy
- The inverse problem
  - Given an MDP and an optimal policy, can we change the values of some weights so that this policy remains optimal?

The Inverse Problem

- The direct problem
  - Given an MDP, compute an optimal policy
- The inverse problem
  - Given an MDP and an optimal policy, can we change the values of some weights so that this policy remains optimal?
- More formally...

Goal

Given an MDP $\mathcal{M}$ and an optimal policy μ_0, compute a constraint K_0 on the weights seen as parameters such that, for any value of the parameters, the policy μ_0 remains optimal

Outline

Outline

The Classical Value Determination Algorithm

- Used by the policy iteration algorithm to compute the optimal policy
- Inputs
  - A Markov decision process $\mathcal{M} = (S, A, Prob, w)$
  - A policy μ
- Output
  - A value function v, associating a value to every state s, i.e., the cost from s to the absorbing state in $\mathcal{M}$ restricted to policy μ

Algorithm (Value Determination)

SOLVE $\{v(s) = w(s, \mu[s]) + \sum_{s' \in S} Prob(s, \mu[s], s') \times v(s')\}_{s \in S \setminus s_n}$

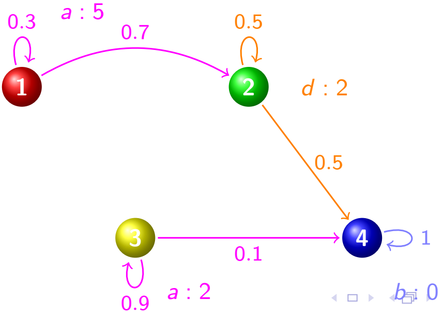
The Classical Value Determination Algorithm: Application

Algorithm (Value Determination)

SOLVE $\{v(s) = w(s, \mu[s]) + \sum_{s' \in S} Prob(s, \mu[s], s') \times v(s')\}_{s \in S \setminus s_n}$

$\mu = \{1 \rightarrow a, \quad 2 \rightarrow d, \quad 3 \rightarrow a\}$

$$v(1) = w(1, a) + 0.3 \times v(1) + 0.7 \times v(2)$$
$$v(2) = w(2, d) + 0.5 \times v(2) + 0.5 \times v(4)$$
$$v(3) = w(3, a) + 0.9 \times v(3) + 0.1 \times v(4)$$
$$v(4) = 0$$



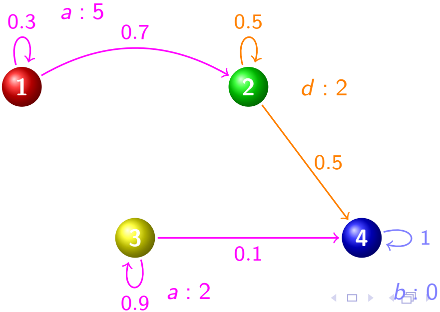
The Classical Value Determination Algorithm: Application

Algorithm (Value Determination)

SOLVE $\{v(s) = w(s, \mu[s]) + \sum_{s' \in S} Prob(s, \mu[s], s') \times v(s')\}_{s \in S \setminus s_n}$

$\mu = \{1 \to a, \quad 2 \to d, \quad 3 \to a\}$

$$
\begin{aligned}
v(1) &= w(1,a) + 0.3 \times v(1) + 0.7 \times v(2) & \\
v(2) &= w(2,d) + 0.5 \times v(2) + 0.5 \times v(4) & = 4 \\
v(3) &= w(3,a) + 0.9 \times v(3) + 0.1 \times v(4) & = 20 \\
v(4) &= 0 &
\end{aligned}
$$



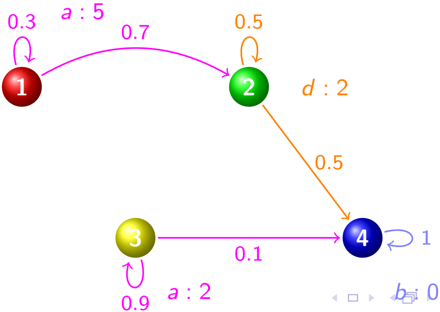
The Classical Value Determination Algorithm: Application

Algorithm (Value Determination)

SOLVE $\{v(s) = w(s, \mu[s]) + \sum_{s' \in S} Prob(s, \mu[s], s') \times v(s')\}_{s \in S \setminus s_n}$

$\mu = \{1 \rightarrow a, \quad 2 \rightarrow d, \quad 3 \rightarrow a\}$

$$\begin{aligned} v(1) &= w(1, a) + 0.3 \times v(1) + 0.7 \times v(2) &&= \frac{78}{7} \\ v(2) &= w(2, d) + 0.5 \times v(2) + 0.5 \times v(4) &&= 4 \\ v(3) &= w(3, a) + 0.9 \times v(3) + 0.1 \times v(4) &&= 20 \\ v(4) &= 0 \end{aligned}$$



The Classical Policy Iteration Algorithm

- Input: A Markov decision process $\mathcal{M} = (S, A, Prob, w)$
- Output: An optimal policy μ
- Principle:
  1. Start with a random policy
  2. Compute the value function, using algorithm *ValueDet*
  3. Choose a strictly better policy, and go to (2) until fixpoint

Algorithm (Policy Iteration)

REPEAT UNTIL FIXPOINT

$v := ValueDet(M, \mu)$

for each $s \in S \setminus s_n$ **DO**

$optimum := v[s]$

for each $a \in e(s)$ **DO**

IF $w(s, a) + \sum_{s' \in S} Prob(s, a, s')v(s') < optimum$ **THEN**

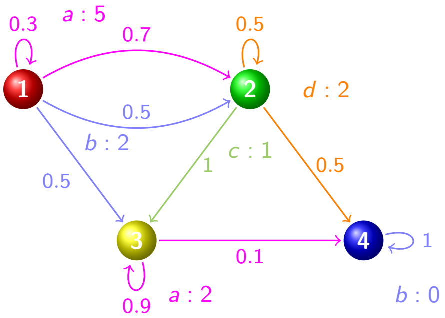
$optimum := w(s, a) + \sum_{s' \in S} Prob(s, a, s')v(s')$

$\mu[s] := a$

The Classical Policy Iteration Algorithm: Application

$v(1) = \frac{197}{7}$
$v(2) = 21$
$v(3) = 20$
$v(4) = 0$

μ :
$1 \rightarrow a$
$2 \rightarrow c$
$3 \rightarrow a$

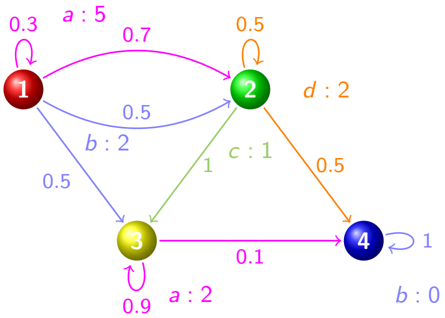


1. We start from an arbitrary policy

The Classical Policy Iteration Algorithm: Application

$v(1) = 14$
$v(2) = 4$
$v(3) = 20$
$v(4) = 0$

μ :
$1 \to b$
$2 \to d$
$3 \to a$

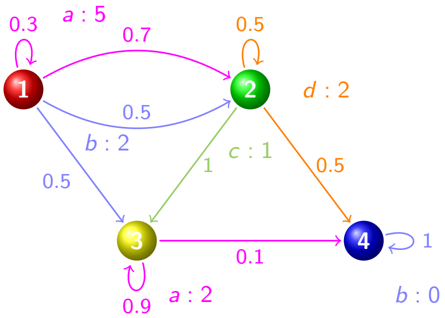


1. We start from an arbitrary policy
2. We improve policy for states 1 and 2

The Classical Policy Iteration Algorithm: Application

$v(1) = \frac{78}{7}$
$v(2) = 4$
$v(3) = 20$
$v(4) = 0$

μ :
$1 \to a$
$2 \to d$
$3 \to a$

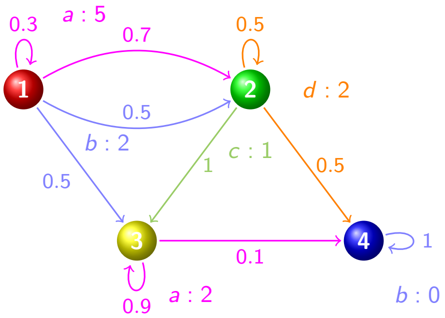


1. We start from an arbitrary policy
2. We improve policy for states 1 and 2
3. We improve policy for state 1

The Classical Policy Iteration Algorithm: Application

$v(1) = \frac{78}{7}$
$v(2) = 4$
$v(3) = 20$
$v(4) = 0$

μ :
$1 \to a$
$2 \to d$
$3 \to a$

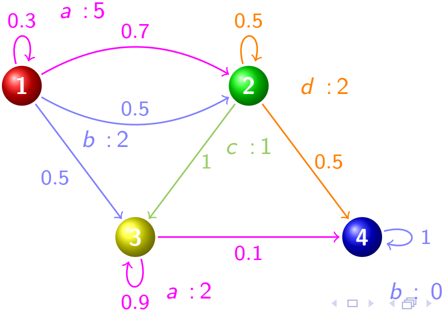


1. We start from an arbitrary policy
2. We improve policy for states 1 and 2
3. We improve policy for state 1
4. Fixpoint is reached: the policy μ is optimal for $\mathcal{M}$

Outline

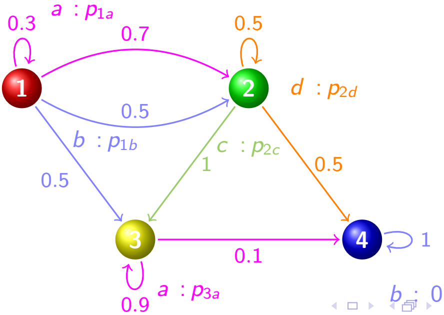
Markov Decision Process

- Markov Decision Process
  - A set of states $S = \{s_1, \ldots, s_n\}$, including one absorbing state s_n
  - A set A of actions
  - A weight function w, associating a cost $w(s, a)$ to every state s and action a
  - A probability function $Prob$, associating a probability to every edge, such that the sum of the probabilities of leaving a state s through action a is equal to 1, i.e., $\sum_{s' \in S} Prob(s, a, s') = 1$



Parametric Markov Decision Process

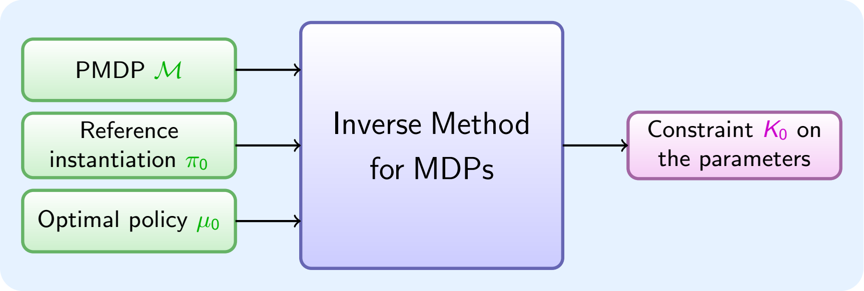
- Markov Decision Process with parametric weights
  - A set of states $S = \{s_1, \ldots, s_n\}$, including one absorbing state s_n
  - A set A of actions
  - A parametric weight function W, associating a parametric cost (i.e., unknown constant) $W(s, a)$ to every state s and action a
  - A probability function $Prob$, associating a probability to every edge, such that the sum of the probabilities of leaving a state s through action a is equal to 1, i.e., $\sum_{s' \in S} Prob(s, a, s') = 1$



Parametric Markov Decision Process: Remarks

- Instantiating a PMDP $\mathcal{M}$ with a valuation π of the parameters gives a (non-parametric) MDP
  - Denoted by $\mathcal{M}[\pi]$
- A PMDP models the behavior of an infinite number of MDPs
- The parametrization of an MDP into a PMDP is similar to the parametrization of a Timed Automaton into a Parametric Timed Automaton

Inputs and Outputs (1/2)

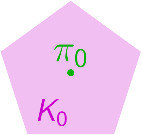


Inputs and Outputs (2/2)

- Inputs
  - A Parametric MDP $\mathcal{M}$
  - A reference instantiation π_0 of all the parameters of $\mathcal{M}$
  - A policy μ_0 optimal for $\mathcal{M}[\pi_0]$

Inputs and Outputs (2/2)

- Inputs
  - A Parametric MDP $\mathcal{M}$
  - A reference instantiation π_0 of all the parameters of $\mathcal{M}$
  - A policy μ_0 optimal for $\mathcal{M}[\pi_0]$
- Output: generalization
  - A constraint K_0 on the parameters such that
    - ⋆ $\pi_0 \models K_0$
    - ⋆ The policy μ_0 is optimal for $\mathcal{M}[\pi]$, for all $\pi \models K_0$



The General Idea

Given a PMDP $\mathcal{M}$, an instantiation π_0 of the parameters, and a policy μ_0 optimal for $\mathcal{M}[\pi_0]$

1. Compute a parametric value function for $\mathcal{M}$ and μ_0, using a parametric version of the value determination algorithm
2. Generate constraints on the parameters of $\mathcal{M}$, using a parametric version of the policy iteration algorithm

The Parametric Value Determination Algorithm

- Straightforward adaptation of the value determination algorithm to the parametric case
- Inputs
  - A parametric Markov decision process $\mathcal{M} = (S, A, Prob, W)$
  - A policy μ
- Output
  - A parametric value function V, associating a parametric value to every state s, i.e., the parametric cost from s to the absorbing state in $\mathcal{M}$ restricted to policy μ

Algorithm (Parametric Value Determination *P-ValueDet*)

SOLVE $\{V(s) = W(s, \mu[s]) + \sum_{s' \in S} Prob(s, \mu[s], s') \times V(s')\}_{s \in S \setminus s_n}$

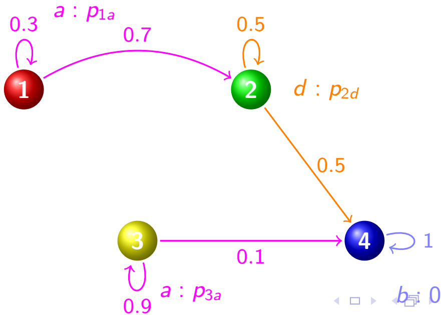
Algorithm *P-ValueDet*: Application

Algorithm (Parametric Value Determination *P-ValueDet*)

SOLVE $\{V(s) = W(s, \mu[s]) + \sum_{s' \in S} Prob(s, \mu[s], s') \times V(s')\}_{s \in S \setminus s_n}$

$\mu = \{1 \rightarrow a, \;\; 2 \rightarrow d, \;\; 3 \rightarrow a\}$

$$V(1) = W(1, a) + 0.3 \times V(1) + 0.7 \times V(2)$$
$$V(2) = W(2, d) + 0.5 \times V(2) + 0.5 \times V(4)$$
$$V(3) = W(3, a) + 0.9 \times V(3) + 0.1 \times V(4)$$
$$V(4) = 0$$



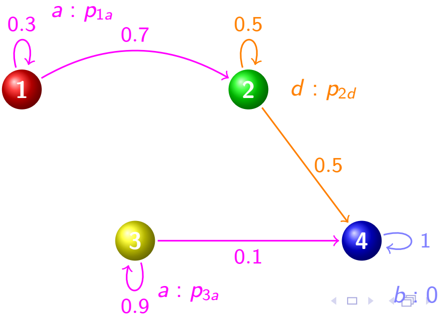
Algorithm *P-ValueDet*: Application

Algorithm (Parametric Value Determination *P-ValueDet*)

SOLVE $\{V(s) = W(s,\mu[s]) + \sum_{s' \in S} Prob(s,\mu[s],s') \times V(s')\}_{s \in S \setminus s_n}$

$\mu = \{1 \to a, \quad 2 \to d, \quad 3 \to a\}$

$$\begin{aligned}
V(1) &= W(1,a) + 0.3 \times V(1) + 0.7 \times V(2) &&= \tfrac{10}{7} \times p_{1a} + 2 \times p_{2d} \\
V(2) &= W(2,d) + 0.5 \times V(2) + 0.5 \times V(4) &&= 2 \times p_{2d} \\
V(3) &= W(3,a) + 0.9 \times V(3) + 0.1 \times V(4) &&= 10 \times p_{3a} \\
V(4) &= 0
\end{aligned}$$



The Algorithm *InverseMethod*

- Inputs
  - A PMDP $\mathcal{M} = (S, A, Prob, W)$
  - An instantiation π_0 of the parameters
  - A policy μ_0 optimal for $\mathcal{M}[\pi_0]$
- Output
  - A constraint K_0 on the parameters solving the inverse problem
- Principle
  - For each state s, for each action a, generate an inequality stating that the optimal policy $\mu_0[s]$ is better than a for s

Algorithm (*InverseMethod*)

$V \ := P\text{-}ValueDet(M, \mu_0)$

$K_0 := True$

FOR EACH $s \in S \setminus \{s_n\}$ **DO**

  FOR EACH $a \in e(s)$ $s.t.$ $a \neq \mu_0[s]$ **DO**

    $K_0 := K_0 \wedge \{W(s,a) + \sum_{s' \in S} Prob(s,a,s')V[s'] \geq V[s]\}$

Properties of the Algorithm *InverseMethod*

Theorem (Correctness)

Given a PMDP $\mathcal{M}$, a reference instantiation π_0 and a policy μ_0 optimal for $\mathcal{M}[\pi_0]$, the constraint K_0 output by the algorithm InverseMethod is such that

- $\pi_0 \models K_0$, *and*
- μ_0 *is optimal for* $\mathcal{M}[\pi]$, *for all* $\pi \models K_0$

Theorem (Termination and complexity)

The algorithm InverseMethod terminates in polynomial time.

Application to Our Example

π_0 :

$p_{1a} = 5 \qquad p_{1b} = 2$

$p_{2c} = 1 \qquad p_{2d} = 2$

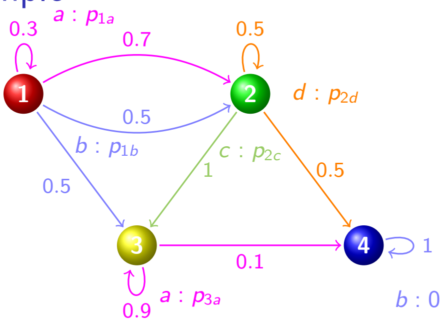
$p_{3a} = 2$

μ_0 :

$1 \rightarrow a$

$2 \rightarrow d$

$3 \rightarrow a$



Application to Our Example

π_0 :

$p_{1a} = 5 \qquad p_{1b} = 2$

$p_{2c} = 1 \qquad p_{2d} = 2$

$p_{3a} = 2$

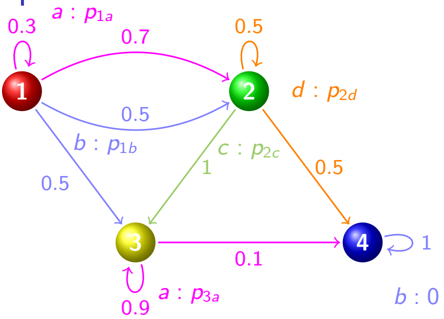
μ_0 :

$1 \rightarrow a$

$2 \rightarrow d$

$3 \rightarrow a$

$$V(1) = \frac{10}{7} \times p_{1a} + 2 \times p_{2d}$$
$$V(2) = 2 \times p_{2d}$$
$$V(3) = 10 \times p_{3a}$$



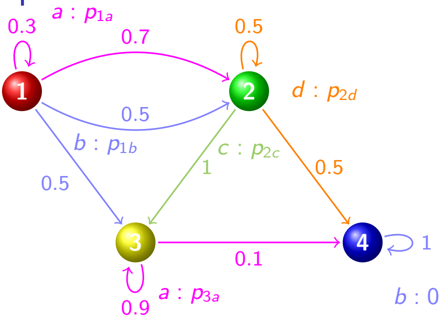
Application to Our Example

π_0 :

$p_{1a} = 5$ $\quad p_{1b} = 2$
$p_{2c} = 1$ $\quad p_{2d} = 2$
$p_{3a} = 2$

μ_0 :

$1 \rightarrow a$
$2 \rightarrow d$
$3 \rightarrow a$



$$V(1) = \frac{10}{7} \times p_{1a} + 2 \times p_{2d}$$
$$V(2) = 2 \times p_{2d}$$
$$V(3) = 10 \times p_{3a}$$

$K_0 = True$

Application to Our Example

π_0 :

$p_{1a} = 5 \qquad p_{1b} = 2$

$p_{2c} = 1 \qquad p_{2d} = 2$

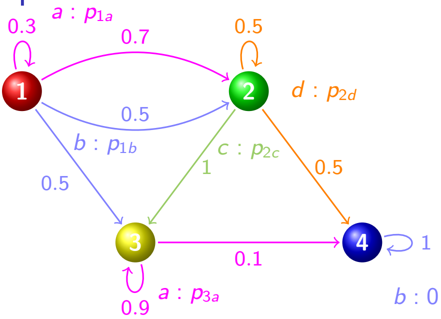
$p_{3a} = 2$

μ_0 :

$1 \rightarrow a$

$2 \rightarrow d$

$3 \rightarrow a$



$$V(1) = \frac{10}{7} \times p_{1a} + 2 \times p_{2d}$$
$$V(2) = 2 \times p_{2d}$$
$$V(3) = 10 \times p_{3a}$$

$K_0 =$

$p_{1b} + \frac{1}{2}V(2) + \frac{1}{2}V(3) \geq V(1)$ %% for 1 and b

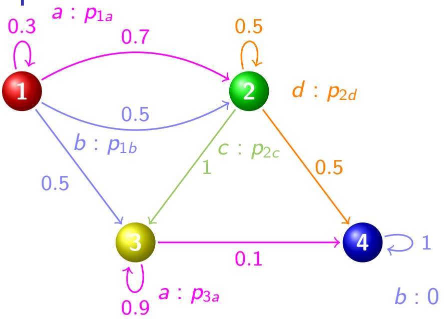
Application to Our Example

π_0 :

$p_{1a} = 5 \qquad p_{1b} = 2$

$p_{2c} = 1 \qquad p_{2d} = 2$

$p_{3a} = 2$



μ_0 :

$1 \rightarrow a$

$2 \rightarrow d$

$3 \rightarrow a$

$$V(1) = \frac{10}{7} \times p_{1a} + 2 \times p_{2d}$$
$$V(2) = 2 \times p_{2d}$$
$$V(3) = 10 \times p_{3a}$$

$K_0 =$

$p_{1b} + \frac{1}{2}V(2) + \frac{1}{2}V(3) \geq V(1)$ %% for 1 and b

$\wedge \quad p_{2c} + V(3) \geq V(2)$ %% for 2 and c

Application to Our Example

π_0 :

$p_{1a} = 5 \qquad p_{1b} = 2$

$p_{2c} = 1 \qquad p_{2d} = 2$

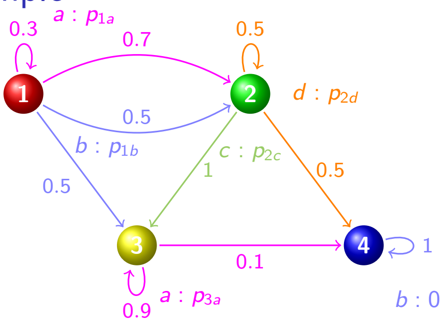
$p_{3a} = 2$

μ_0 :

$1 \rightarrow a$

$2 \rightarrow d$

$3 \rightarrow a$



$$V(1) = \frac{10}{7} \times p_{1a} + 2 \times p_{2d}$$
$$V(2) = 2 \times p_{2d}$$
$$V(3) = 10 \times p_{3a}$$

$K_0 =$

$$p_{1b} + 5p_{3a} \geq \frac{10}{7} p_{1a} + p_{2d}$$
$$\wedge \quad p_{2c} + 10p_{3a} \geq 2p_{2d}$$

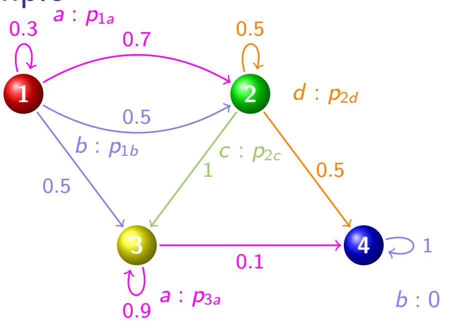
Application to Our Example

π_0 :

$p_{1a} = 5 \qquad p_{1b} = 2$

$p_{2c} = 1 \qquad p_{2d} = 2$

$p_{3a} = 2$



μ_0 :

$1 \to a$

$2 \to d$

$3 \to a$

$$V(1) = \frac{10}{7} \times p_{1a} + 2 \times p_{2d}$$
$$V(2) = 2 \times p_{2d}$$
$$V(3) = 10 \times p_{3a}$$

$$K_0 = \quad p_{1b} + 5p_{3a} \geq \frac{10}{7} p_{1a} + p_{2d} \quad \wedge \quad p_{2c} + 10p_{3a} \geq 2p_{2d}$$

- Application: maximization of, e.g., p_{2d}
  - By instantiating all parameters except p_{2d} within K_0, we get $p_{2d} \leq \frac{34}{7}$
  - We can thus maximize p_{2d} to $\frac{34}{7}$ so that μ_0 remains optimal

Outline

Implementation

- IMPRATOR: program written in OCaml
  - IMPRATOR: "Inverse Method for Policy with Reward AbstracT BehaviOR"
  - 4000 lines of code
  - 2 man-months of work
- Features
  - Very intuitive input syntax
  - Solves the direct problem for (non-parametric) MDPs
  - Solves the inverse problem for parametric MDPs
- IMPRATOR will be available on its Web page
  - http://www.lsv.ens-cachan.fr/~andre/ImPrator
  - Coming (very) soon!

Outline

Final Remarks (1/2)

- Generalization method
  - Modeling of a system with a parametric Markov decision process $\mathcal{M}$
  - Starting with an instantiation π_0 of the parameters, as well a policy μ_0 optimal for $\mathcal{M}[\pi_0]$, we generate a constraint K_0 on the parameters guaranteeing that μ_0 is optimal for $\mathcal{M}[\pi]$, for any $\pi \models K_0$
- Advantages
  - Useful to optimize costs of systems, e.g., hardware devices
  - Powerful even on fully parametrized big systems
    - All case studies terminated in less than 1 second
- Applications
  - Real time systems
  - Hardware verification

Final Remarks (2/2)

- Other frameworks for the inverse method
  - Parametric Timed Automata [ACEF09]
    - Tool IMITATOR [And09]
  - Max–Plus Algebra [AF09]
    - Computation of the maximal circuit mean in a directed weighted graph
    - Tool under development

- Future works
  - Prove that the generated K_0 is maximal
    - If μ_0 is an optimal policy for $M[\pi]$, then $\pi \models K_0$
  - Handle MDPs with 2 kinds of weights
    - Example: (1) power consumption and (2) number of lost requests
    - Application: dynamic power management [PBBDM98]

References I

Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg.
An inverse method for parametric timed automata.
International Journal of Foundations of Computer Science, 2009.
To appear.

É. André and L. Fribourg.
An inverse method for policy-iteration based algorithms.
In *INFINITY '09*, August 2009.

Étienne André.
IMITATOR: A tool for synthesizing constraints on timing bounds of timed automata.
In *ICTAC'09*, LNCS. Springer, August 2009.
To appear.

R. Bellman.
A Markov decision process.
Journal of Mathematical Mechanics, 6:679–684, 1957.

R. A. Howard.
Dynamic Programming and Markov Processes.
John Wiley and Sons, Inc., 1960.

References II

J. Kemeny, H. Mirkil, J. Snell, and G. Thompson.
Finite mathematical structures.
Prentice-Hall, Englewood Cliffs, N.J., 1959.

G. A. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli.
Policy optimization for dynamic power management.
In *DAC '98*, pages 182–187, New York, NY, USA, 1998. ACM.