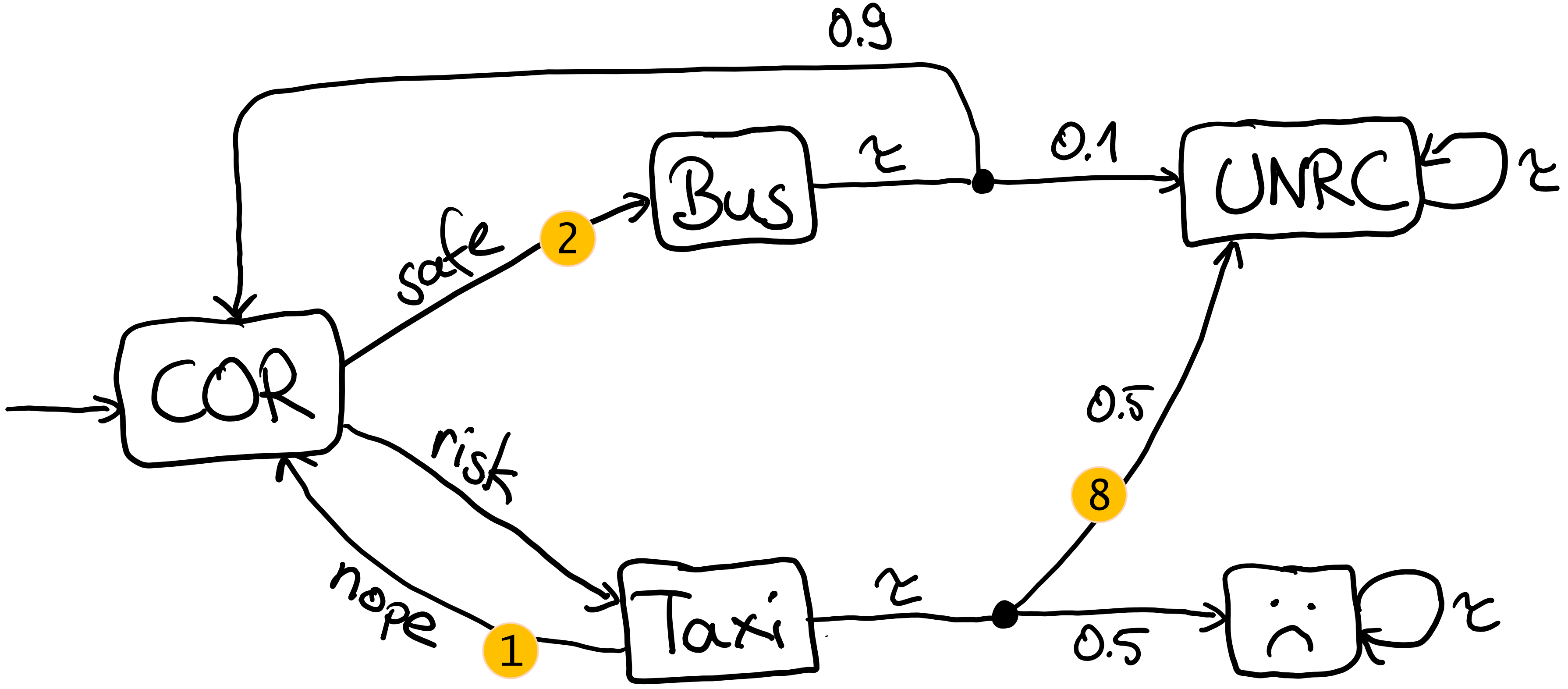
Formal Approaches to Decision-Making under Uncertainty

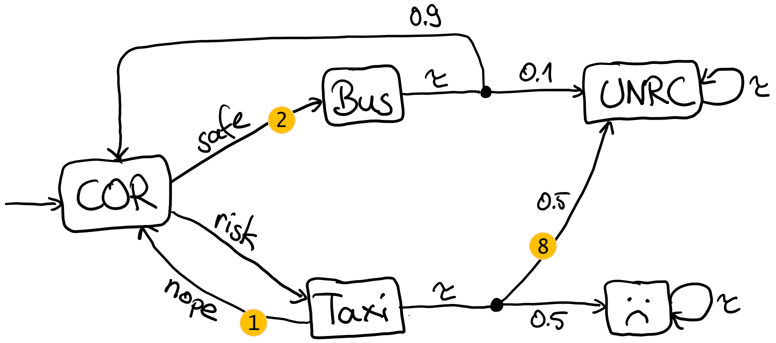
Lecture 3-2: Algorithms for MDPs

Arnd Hartmanns
Formal Methods and Tools
UNIVERSITY OF TWENTE

Recall MDPs:



Value Iteration



$T: S \to 2^{A \times \mathrm{Dist}(S)}$

$\langle S, A, T, s_I \rangle$

Adapt **value iteration** to MDP and $\mathrm{P}_{opt}(\Diamond^{\leq b}\, G)$:

1. Make states in G absorbing.
2. Iterate:

$$x_0[s] = 1 \text{ if } s \in G \text{ else } 0$$

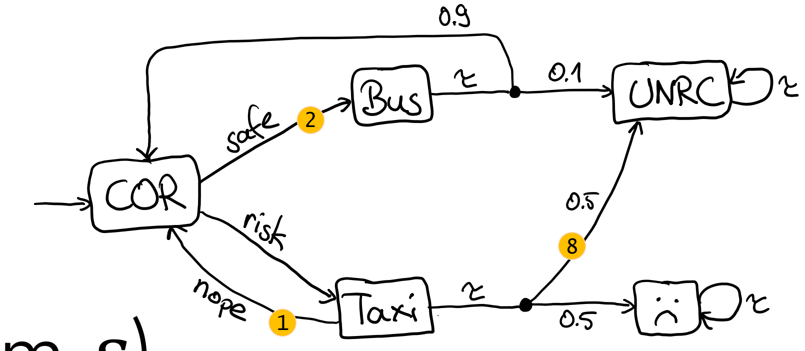
$$x_i[s] = opt_{s \xrightarrow{a} \mu} \sum_{s'} \mu(s') \cdot x_{i-1}[s']$$

Example:

$G = \{UNRC\}$

i =	0	1	2	3	4	5	
COR	0	0	0.5	0.5	0.55	0.55	
Bus	0	0.1	0.1	0.55	0.55	0.595	
Taxi	0	0.5	0.5	0.5	0.5	0.55	...
UNRC	1	1	1	1	1	1	
:(	0	0	0	0	0	0	

Value Iteration



Also for MDP:

Fact 1: $\lim_{i\to\infty} x_i[s] = \mathrm{P}_{opt}(\diamond\, G)$ from s

Fact 2: the vector $\boldsymbol{x}$ (where $x[s] = \mathrm{P}_{opt}(\diamond\, G)$ from s)

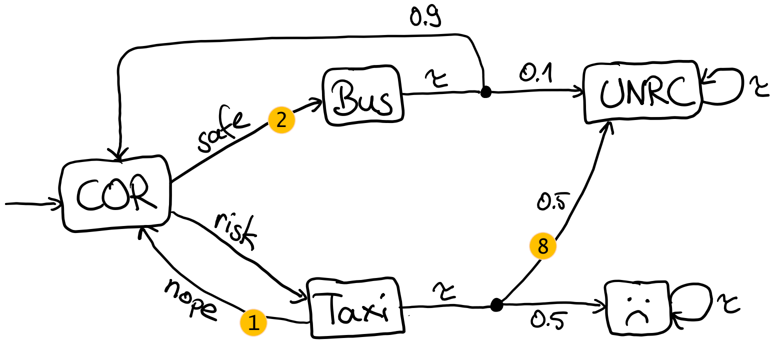
is the *least fixed point* of the Bellman operator

$$x_i[s] = 1 \text{ if } s \in G \text{ else } opt_{s \xrightarrow{a} \mu} \textstyle\sum_{s'} \mu(s') \cdot x_{i-1}[s']$$

→ unbounded VI: same convergence problem as for DTMC

→ fixed point: unique if no *end components*

Value Iteration



Adaptation to $\mathrm{R}_{opt}(\Diamond\, G)$ just like for DTMC:

1. Precompute S_1 states where $\mathrm{P}_{\overline{opt}}(\Diamond\, G) = 1$ with $\overline{\max} = \min$ and $\overline{\min} = \max$
2. If $s_I \notin S_1$: return ∞; otherwise, iterate:

   $x_0[s] = 0$ if $s \in S_1$ else ∞

   $x_i[s] = 0$ if $s \in G$ else

$$opt_{s \xrightarrow{a} \mu} \sum_{s'} \mu(s, s') \cdot (R(s, \langle a, \mu \rangle, s') + x_{i-1}[s])$$

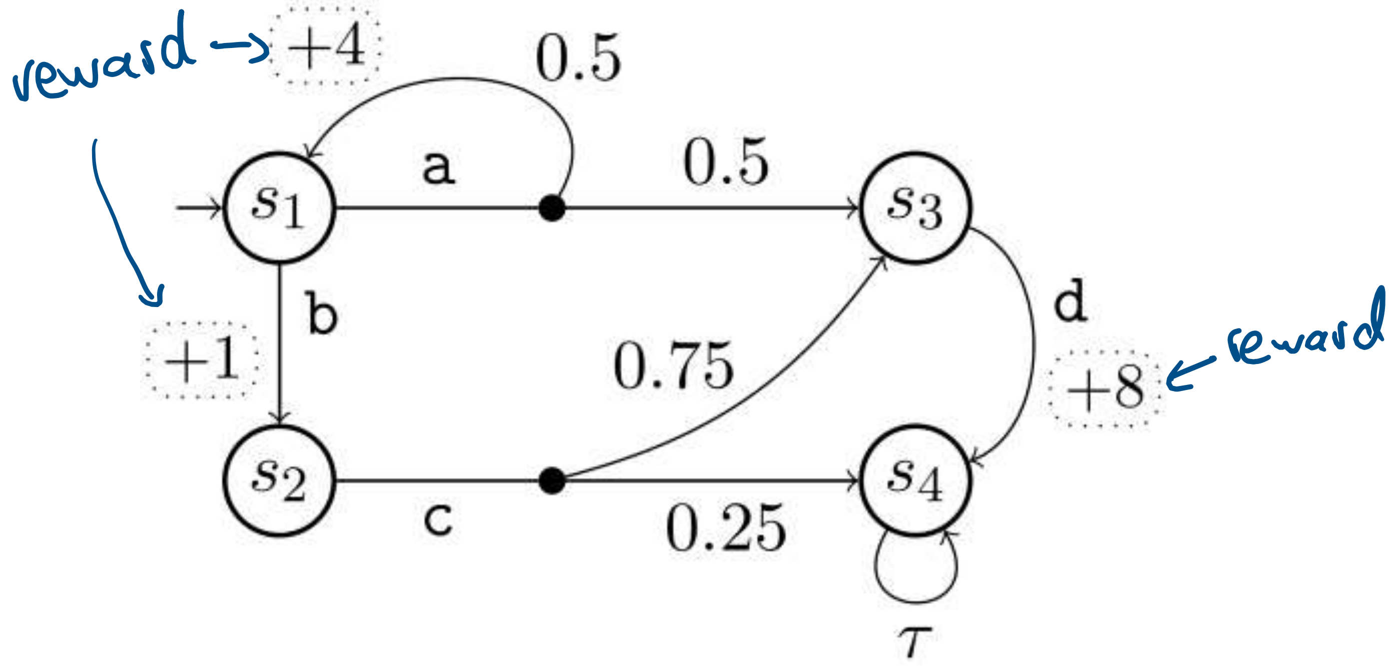
Example:

Exercise V

For the MDP below,

a) use value iteration to compute $\mathrm{P}_{\max}(\Diamond^{\leq 3}\{s_3\})$
and give the corresponding optimal (step-positional) scheduler;

b) use value iteration to compute $\mathrm{R}_{\min}(\Diamond\{s_4\})$
and give the corresponding optimal (memoryless) scheduler.



Policy iteration (PI) or Howard's algorithm

1. Pick some scheduler $\mathcal{S}$
2. Compute vector $\boldsymbol{x}$ where $x[s] = \mathrm{P}_{opt}(\diamond\, G)$ from s for $M|_{\mathcal{S}}$
3. Update scheduler where it is sub-optimal:

$$\mathcal{S}'(s) \coloneqq \arg opt_{s \xrightarrow{a} \mu} \sum_{s'} \mu(s') \cdot x[s']$$

4. If $\mathcal{S}' \neq \mathcal{S}$, then set $\mathcal{S} \coloneqq \mathcal{S}'$ and go to 2.

→ How to implement step 2?

...and analogously for expected rewards.

Linear programming (LP)

Find a vector $\mathbf{x}$
that maximizes $\mathbf{c}^T\mathbf{x}$
subject to $A\mathbf{x} \leq \mathbf{b}$
and $\mathbf{x} \geq \mathbf{0}$.

Linear programming

48 languages

Read Edit View history

...oadcast programming.

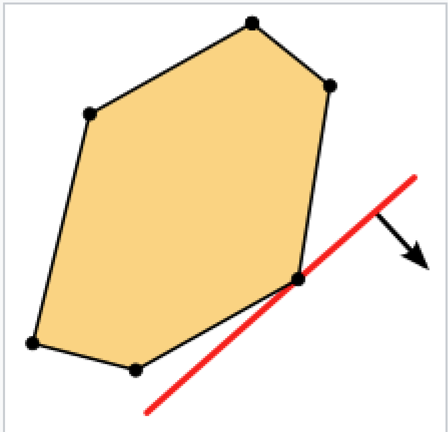
...a method to achieve the best outcome (such as ... equirements are represented by linear ... tical programming (also known as mathematical ...

...ation of a linear objective function, subject to ... n is a convex polytope, which is a set defined as ... fined by a linear inequality. Its objective function is a real-valued affine (linear) function defined on this polyhedron. A linear programming algorithm finds a point in the polytope where this function has the smallest (or largest) value if such a point exists.

Linear programs are problems that can be expressed in canonical form as

$$\begin{array}{ll} \text{Find a vector} & \mathbf{x} \\ \text{that maximizes} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq \mathbf{0}. \end{array}$$

Here the components of $\mathbf{x}$ are the variables to be determined, $\mathbf{c}$ and $\mathbf{b}$ are given vectors (with $\mathbf{c}^T$ indicating that the coefficients of $\mathbf{c}$ are used as a single-row matrix for the purpose of forming the matrix product), and A is a given matrix. The function whose value is to be maximized or minimized ($\mathbf{x} \mapsto \mathbf{c}^T\mathbf{x}$ in this case) is called the



A pictorial representation of a simple linear program with two variables and six inequalities. The set of feasible solutions is depicted in yellow and forms a polygon, a 2-dimensional polytope. The optimum of the linear cost function is where the red line intersects the polygon. The red line is a level set of the cost function, and the arrow indicates the direction in

Linear Programming

Many commercial and open-source LP solvers exist:

Table 2: Available LP solvers ("intr" = interior point)

solver	version	license	exact/fp	parallel	algorithms	mcsta	Storm
CPLEX[3]	22.10	academic	fp	yes	intr + simplex	yes	no
COPT[4]	5.0.5	academic	fp	yes	intr + simplex	yes	no
Gurobi [24]	9.5	academic	fp	yes	intr + simplex	yes	yes
GLPK[5]	4.65	GPL	fp	no	intr + simplex	no	yes
Glop[6]	9.4.1874	Apache	fp	no	simplex only	yes	no
HiGHS[7]	1.2.2	MIT	fp	yes	intr + simplex	yes	no
lp_solve[8]	5.5.2.11	LGPL	fp	no	simplex only	yes	no
Mosek[9]	10.0	academic	fp	yes	intr + simplex	yes	no
SoPlex [23]	6.0.1	academic	both	no	simplex only	no	yes
Z3 [40]	4.8.13	MIT	exact	no	simplex only	no	yes

Linear Programming

Encode the MDP transition constraints as a linear program:

find a vector $\boldsymbol{x} = (x[s])_{s \in S \setminus G}$ with $\forall s \in S \setminus G\colon 0 \leq x_s \leq 1$

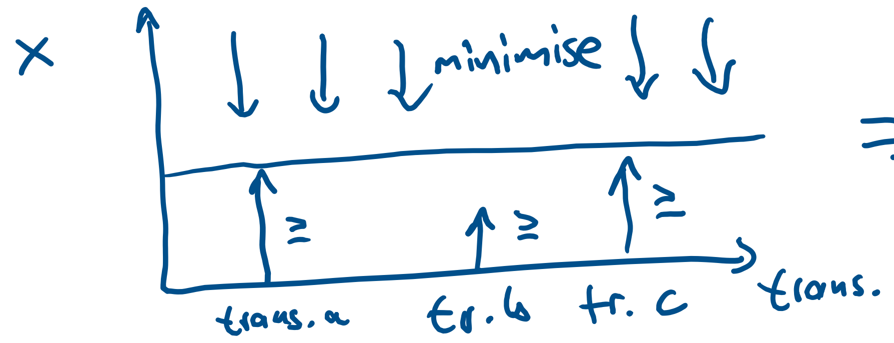
that minimises $\sum_{s \in S \setminus G} x[s]$

subject to

$$x[s] \geq \sum_{s' \in S \setminus G} \mu(s') \cdot x[s'] + \sum_{s_g \in G} \mu(s_g)$$

for all $s \in S \setminus G$, $s \xrightarrow{a} \mu$

Find a vector $\mathbf{x}$
that maximizes $\mathbf{c}^T\mathbf{x}$
subject to $A\mathbf{x} \leq \mathbf{b}$
and $\mathbf{x} \geq \mathbf{0}$.



⇒ max over the transitions

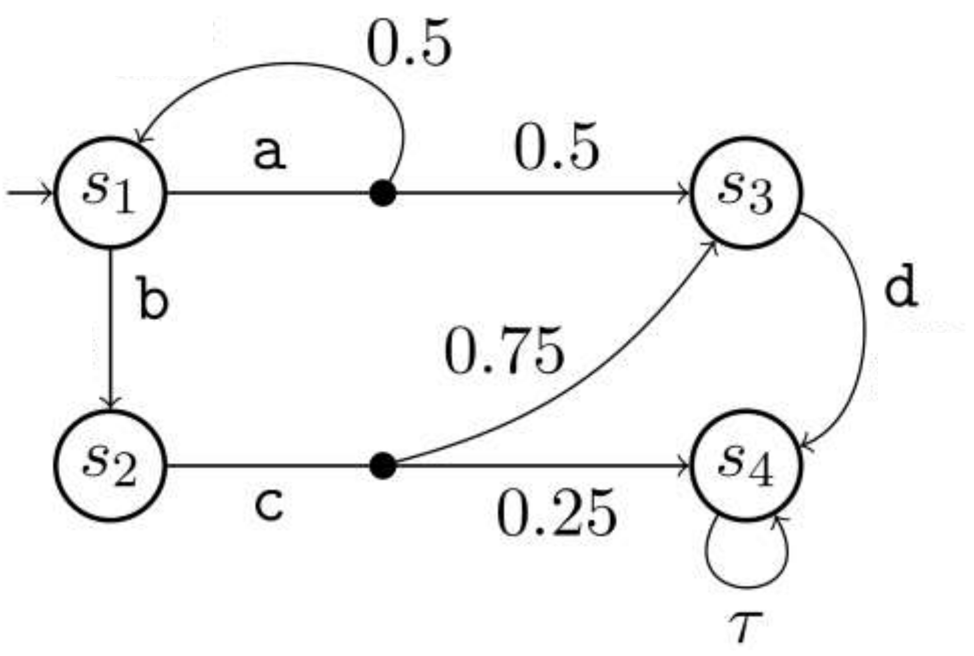
....and analogously for min and expected rewards.

Exercise VI

For the MDP below,

a) use policy iteration to compute $\mathrm{P}_{\max}(\diamond^{\leq 3}\{s_3\})$, documenting the intermediate schedulers that you evaluate, and

b) give the linear program for the same problem.



Model Checking Algorithms

Table 2: Available LP solvers ("intr" = interior point)

solver	version	license	exact/fp	parallel	algorithms	mcsta	Storm
CPLEX[3]	22.10	academic	fp	yes	intr + simplex	yes	no
COPT[4]	5.0.5	academic	fp	yes	intr + simplex	yes	no
Gurobi [24]	9.5	academic	fp	yes	intr + simplex	yes	yes
GLPK[5]	4.65	GPL	fp	no	intr + simplex	no	yes
Glop[6]	9.4.1874	Apache	fp	no	simplex only	yes	no
HiGHS[7]	1.2.2	MIT	fp	yes	intr + simplex	yes	no
lp_solve[8]	5.5.2.11	LGPL	fp	no	simplex only	yes	no
Mosek[9]	10.0	academic	fp	yes	intr + simplex	yes	no
SoPlex [23]	6.0.1	academic	both	no	simplex only	no	yes
Z3 [40]	4.8.13	MIT	exact	no	simplex only	no	yes

Complexity:

VI: exponential

PI: exponential

LP: exponential with simplex algorithm
polynomial with interior-point/barrier methods

Practical performance:

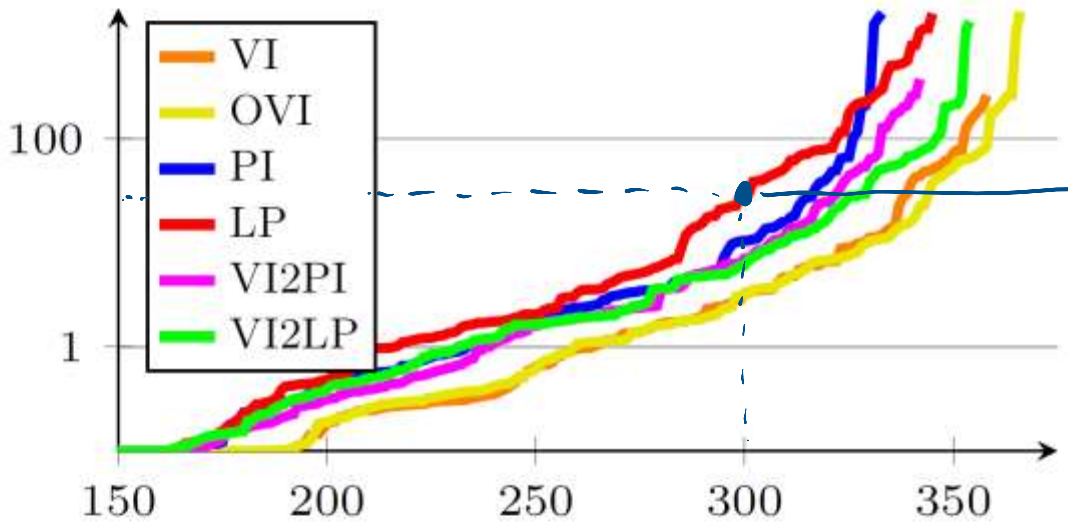
VI: usually fastest

PI: good

LP: depends on solver – ranges from *quite slow* to *like PI*

(Gurobi and COPT currently best)

Model Checking Algorithms



300th-slowest benchmark for LP took ~70 s (difficult with log scale...)

→ ordered by runtime per series (algorithm)

not cumulative!

Practical performance:

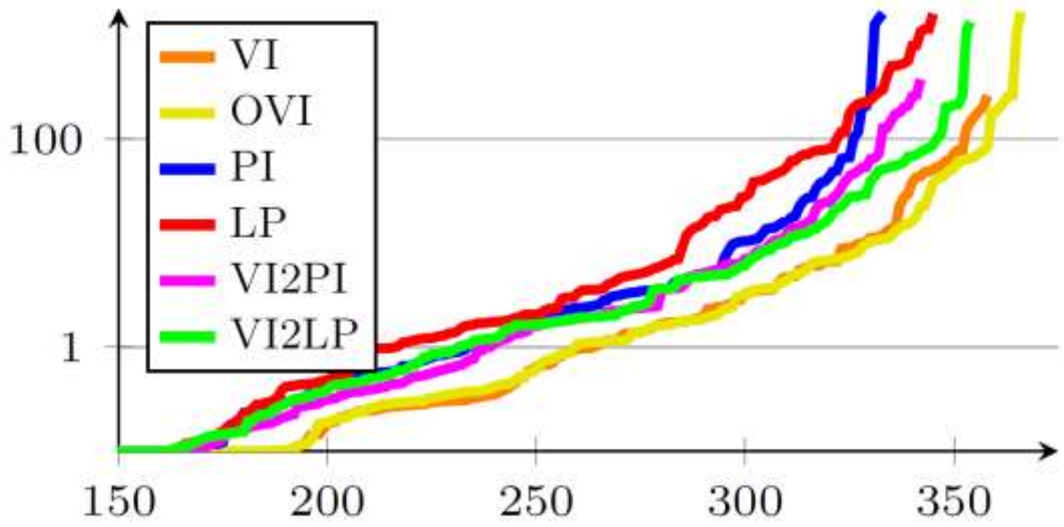
VI: usually fastest

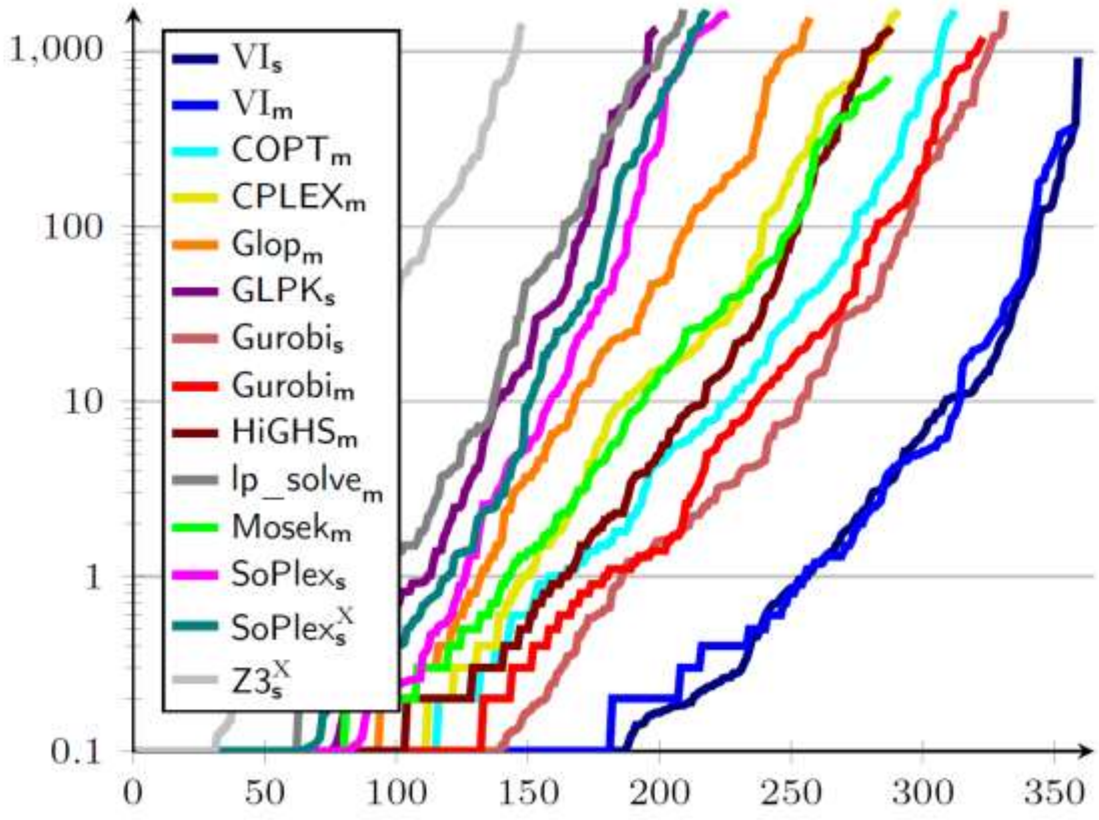
PI: good

LP: depends on solver – ranges from *quite slow* to *like PI*

(Gurobi and COPT currently best)

Model Checking Algorithms





Practical performance:

VI: usually fastest

PI: good

LP: depends on solver – ranges from *quite slow* to *like PI*

(Gurobi and COPT currently best)

Exact and Sound Algorithms

Let $v = \mathrm{P}_{opt}(\diamond\, G)$ be the true, unknown value of interest.

Exact algorithms:

Compute exact result $\bar{v}$ so that $\bar{v} = v$;

need arbitrary-precision rational numbers

Sound algorithms:

Obtain approximate result $\bar{v}$

with $|\bar{v} - v| \leq \epsilon$ (absolute error)

or $|\bar{v} - v|/v \leq \epsilon$ (relative error)

using finite-precision floating-point arithmetic

Exact and Sound Algorithms

Non-exact implementations:

VI: unsound – lack of (efficient) stopping criterion

II: sound – modulo floating-point errors

PI: unsound unless precise – even if DTMC solved ϵ-correctly

LP: unsound – non-exact solvers give no guarantees

Floating-point computations:

finite precision → rounding errors at every step

Exact (arbitrary-precision rational) implementations:

slow, do not scale to large models

Homework

Second step to pass this course:

Try to solve Exercises I to VI from the two slide sets of today, and send your solutions (photos or scans) to Arnd by email.

Try to get as far as you can.

a.hartmanns@utwente.nl