Monte Carlo Tree Search with Advice

Presented by : Debraj Chakraborty

ULB

December 20, 2022

Thesis jury: Emmanuel Filiot
Gilles Geeraerts
Jan Křetínský
Kim G. Larsen

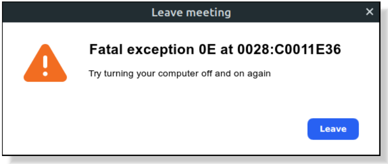
Supervisor: Jean-François Raskin

Introduction

- Bugs : Flaw in the software design causing incorrect results

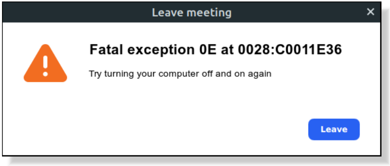
Introduction

- Bugs : Flaw in the software design causing incorrect results



Introduction

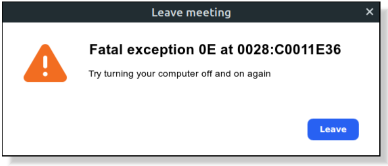
- Bugs : Flaw in the software design causing incorrect results



- Software is everywhere

Introduction

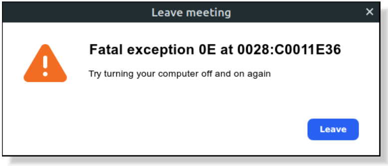
- Bugs : Flaw in the software design causing incorrect results



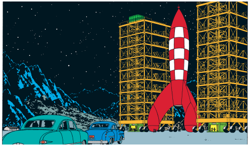
- Software is everywhere

Introduction

- Bugs : Flaw in the software design causing incorrect results

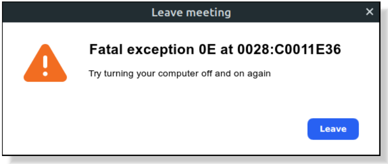


- Software is everywhere

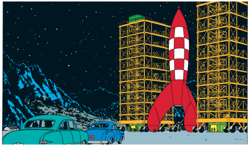


Introduction

- Bugs : Flaw in the software design causing incorrect results

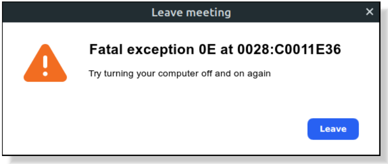


- Software is everywhere

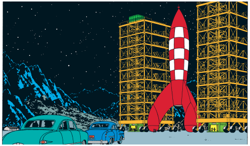


Introduction

- Bugs : Flaw in the software design causing incorrect results



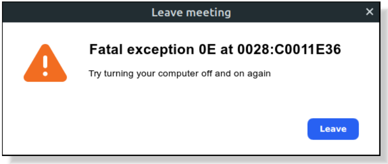
- Software is everywhere



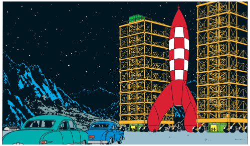
- Reliability of applications depends on the correctness of software

Introduction

- Bugs : Flaw in the software design causing incorrect results



- Software is everywhere



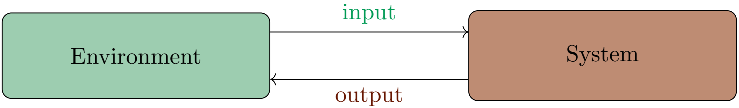

- Reliability of applications depends on the correctness of software
- Bugs can have fatal and costly consequences

Reactive systems

- Continuous interaction between system and environment

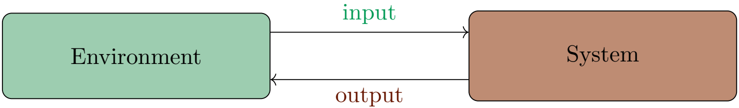
Reactive systems

- Continuous interaction between system and environment
- System receives inputs from the environment and reacts by producing outputs



Reactive systems

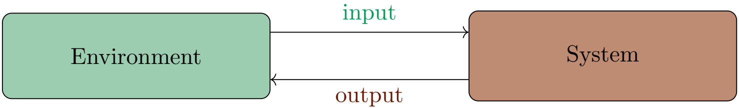
- Continuous interaction between system and environment
- System receives inputs from the environment and reacts by producing outputs



- Reactive systems may need to respect some specific properties

Reactive systems

- Continuous interaction between system and environment
- System receives inputs from the environment and reacts by producing outputs



- Reactive systems may need to respect some specific properties
- Special methods needed to verify that they are bug-free

Verification

Input :

- Model M of the system
- Formal specification φ to describe the property

Output : check all possible executions of M satisfy φ :

$$M \models \varphi$$

Verification

Input :

- Model M of the system
- Formal specification φ to describe the property

Output : check all possible executions of M satisfy φ :

$$M \models \varphi$$

Checkable properties

- Safety : unwanted behaviour never happen.
- Liveness : desired behaviour eventually happen.
- Quantitative properties : energy consumption, cost etc

Synthesis

- Automatic design of a system from the specification.

Synthesis

- Automatic design of a system from the specification.

Input :

- Model M of the system
- Formal specification φ to describe the property

Output :

- Model M such that $M \models \varphi$,
- or No if no model exists.

- More difficult than verification
- 2-player game between system and environment
- Synthesis a strategy for the controller

Different approaches for verification and synthesis

Different approaches for verification and synthesis

Formal methods

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

☺ Strong guarantees ☹ Does not scale to larger systems

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

☺ Strong guarantees ☹ Does not scale to larger systems

Learning-based methods

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

☺ Strong guarantees ☹ Does not scale to larger systems

Learning-based methods

- Heuristic search : simulate possible behaviours of the model

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

☺ Strong guarantees ☹ Does not scale to larger systems

Learning-based methods

- Heuristic search : simulate possible behaviours of the model
- Machine learning : train a neural network

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

☺ Strong guarantees ☹ Does not scale to larger systems

Learning-based methods

- Heuristic search : simulate possible behaviours of the model
- Machine learning : train a neural network
- Not a complete method

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

☺ Strong guarantees ☹ Does not scale to larger systems

Learning-based methods

- Heuristic search : simulate possible behaviours of the model
- Machine learning : train a neural network
- Not a complete method
- Does not suffer from state explosion

Different approaches for verification and synthesis

Formal methods

- Model checking : systematic check of the property in all states of the model
- Exact efficient algorithms
- State explosion : number of states exceeds available memory

☺ Strong guarantees ☹ Does not scale to larger systems

Learning-based methods

- Heuristic search : simulate possible behaviours of the model
- Machine learning : train a neural network
- Not a complete method
- Does not suffer from state explosion

☹ Weaker guarantees ☺ Scales to larger systems

Objective of the thesis

Hybrid algorithms that scales for large systems and provides guarantees

Objective of the thesis

Hybrid algorithms that scales for large systems and provides guarantees

Outline of the presentation

Objective of the thesis

Hybrid algorithms that scales for large systems and provides guarantees

Outline of the presentation

- Markov decision process : model to represent systems

Objective of the thesis

Hybrid algorithms that scales for large systems and provides guarantees

Outline of the presentation

- Markov decision process : model to represent systems
- Monte Carlo tree search : simulation-based heuristic search algorithm

Objective of the thesis

Hybrid algorithms that scales for large systems and provides guarantees

Outline of the presentation

- Markov decision process : model to represent systems
- Monte Carlo tree search : simulation-based heuristic search algorithm
- Monte Carlo tree search with symbolic advice : augmenting MCTS with techniques from formal methods

Objective of the thesis

Hybrid algorithms that scales for large systems and provides guarantees

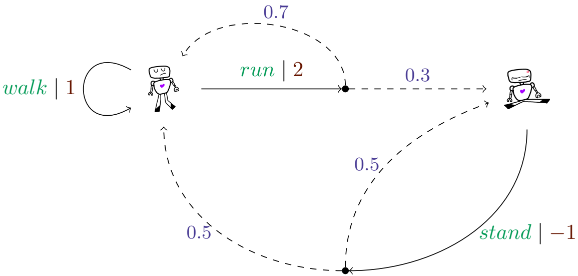
Outline of the presentation

- Markov decision process : model to represent systems
- Monte Carlo tree search : simulation-based heuristic search algorithm
- Monte Carlo tree search with symbolic advice : augmenting MCTS with techniques from formal methods
- Monte Carlo tree search with neural advice : using neural network to imitate and replace the symbolic advice

Markov Decision Process

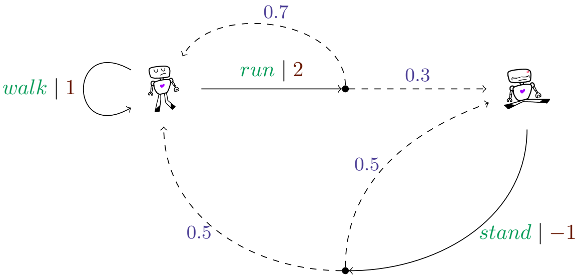
- Controller taking decisions
- Stochastic model of the environment
- Reward as consequence of a decision
- Objective : Synthesis a controller strategy to maximize the reward

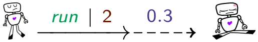
Markov Decision Process



- Path : 
- Reward : 0

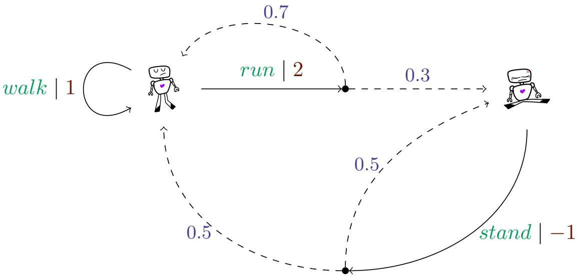
Markov Decision Process

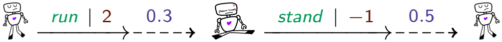


- Path : 

- Reward : 2

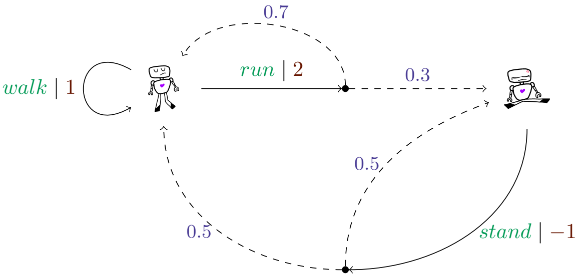
Markov Decision Process



- Path : 

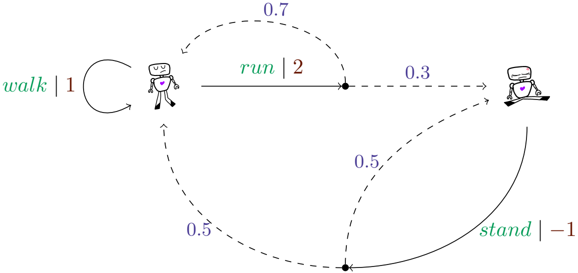
- Reward : 1

Markov Decision Process



- Path : 

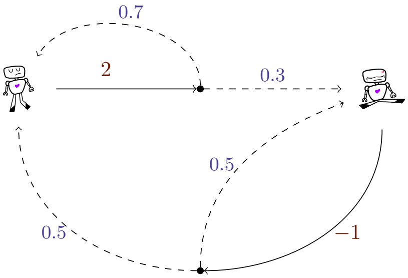
- Reward : 1



- Strategy : Finite paths → Actions

$\{$  $\mapsto$ *run*,  $\mapsto$ *stand* $\}$

Markov Decision Process

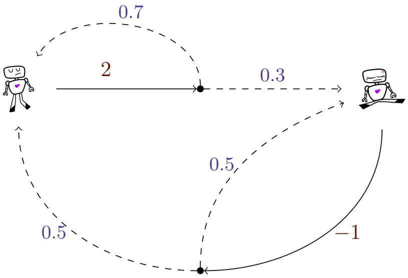


- Strategy : Finite paths $\rightarrow$ Actions

$\Big\{$  $\mapsto$ *run*,  $\mapsto$ *stand* $\Big\}$

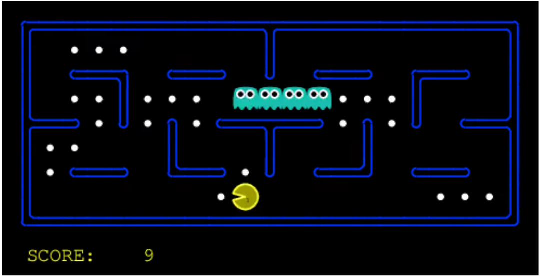
- Fixing a Strategy creates a Markov chain

Markov Decision Process



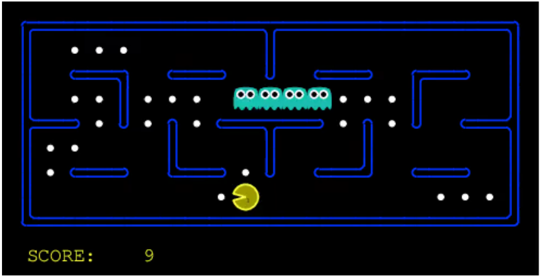
- Finite-horizon reward $\mathrm{Val}_H(s,\sigma) = \mathbb{E}_\sigma\left[\mathrm{Reward}(p)\right]$ for path p of length H in the Markov chain
- Infinite-horizon average reward $\mathrm{Val}(s,\sigma) = \lim_{H\to\infty} \frac{1}{H}\mathrm{Val}_H(s,\sigma)$
- Optimal strategy $\arg\max_\sigma \mathrm{Val}(s,\sigma)$

Example: Pac-Man



- Controller: Pac-Man
- Probabilistic model of ghosts
- States: position of every agent, what food is left
- Actions: Pac-Man moves
- Stochastic transitions: ghost moves

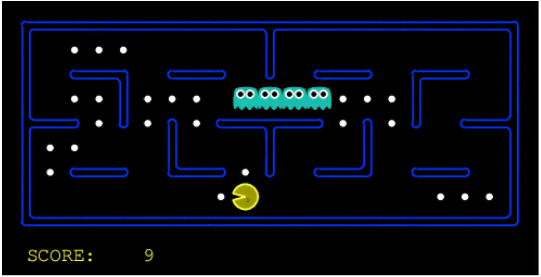
Example: Pac-Man



- Controller: Pac-Man
- Probabilistic model of ghosts
- Reward for eating food
- Large penalty for losing

- States: position of every agent, what food is left
- Actions: Pac-Man moves
- Stochastic transitions: ghost moves

Example: Pac-Man

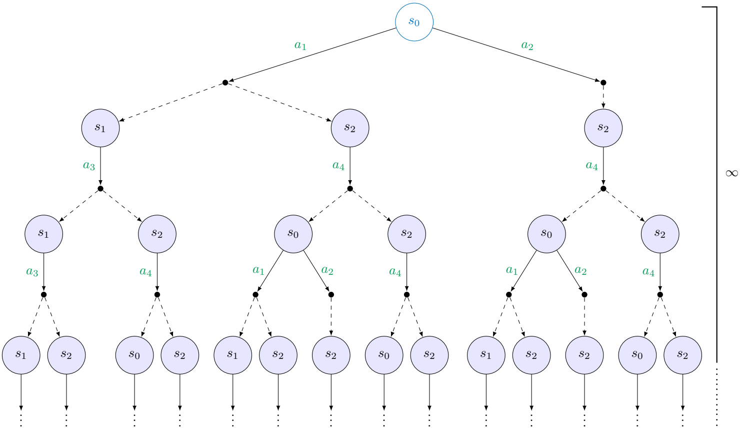


- Controller: Pac-Man
- Probabilistic model of ghosts
- Reward for eating food
- Large penalty for losing

- States: position of every agent, what food is left
- Actions: Pac-Man moves
- Stochastic transitions: ghost moves
- Large MDP: $\sim 10^{16}$ states

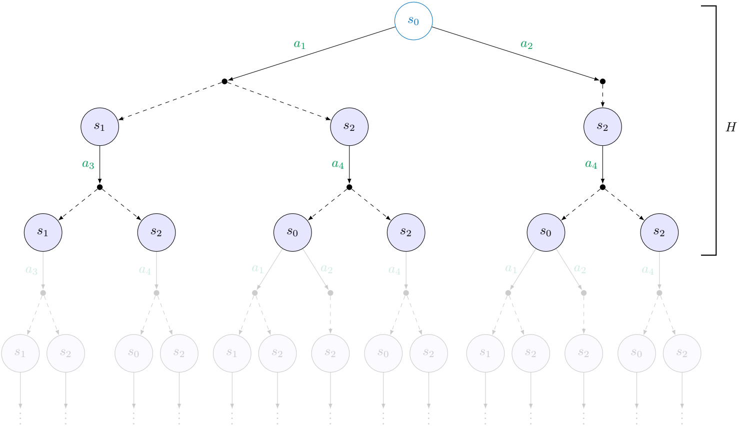
Receding horizon control

Receding horizon control



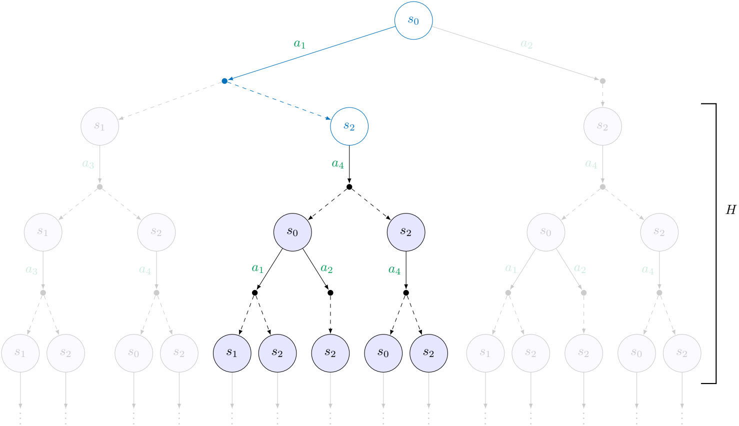
- Unfolding of the MDP

Receding horizon control



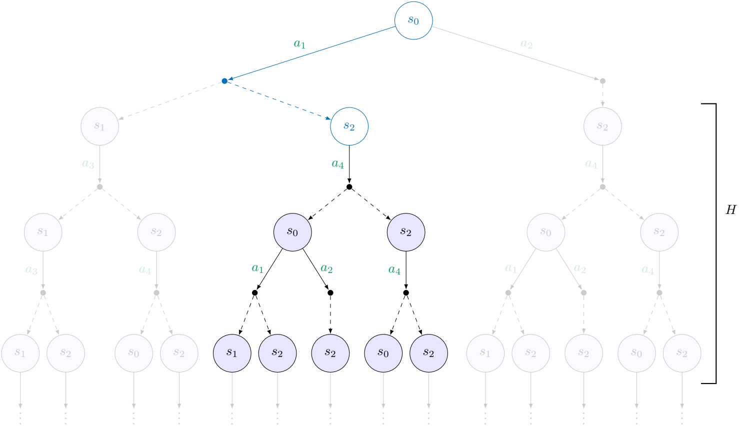
- Unfolding of the MDP
- Optimize for total reward with a sliding window of H

Receding horizon control



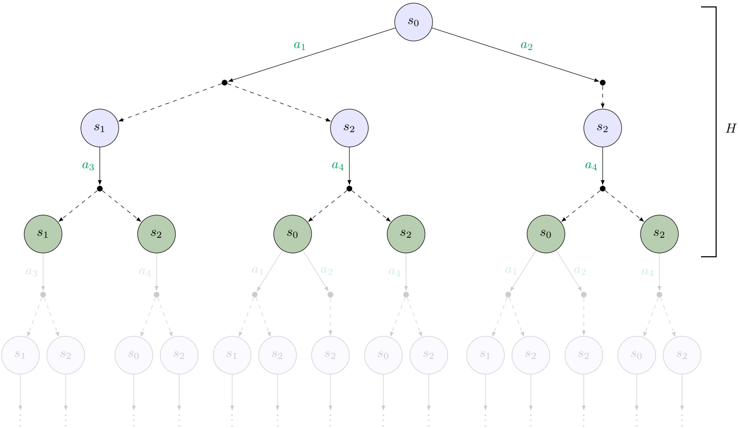
- Unfolding of the MDP
- Optimize for total reward with a sliding window of H

Receding horizon control



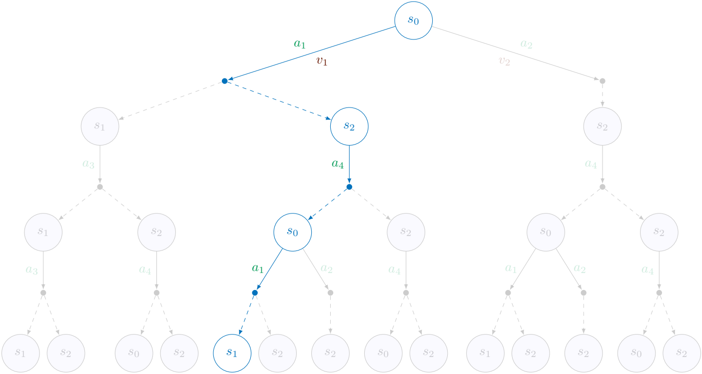
- Unfolding of the MDP
- Optimize for total reward with a sliding window of H
- H large enough $\rightsquigarrow$ optimal strategy

Receding horizon control



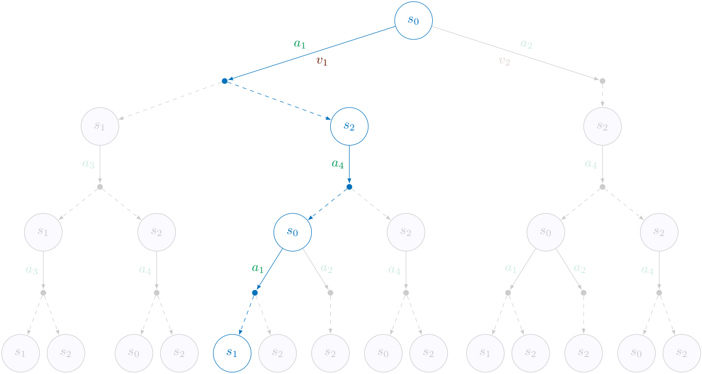
- Unfolding of the MDP
- Optimize for total reward with a sliding window of H
- H large enough $\rightsquigarrow$ optimal strategy
- H not large enough $\rightsquigarrow$ terminal reward using a heuristic function to estimate long-term behaviour

Heuristic search



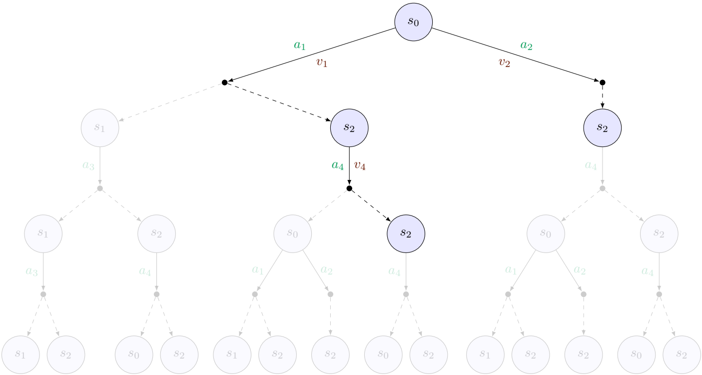
- Large unfolding $\rightsquigarrow$ heuristics

Heuristic search



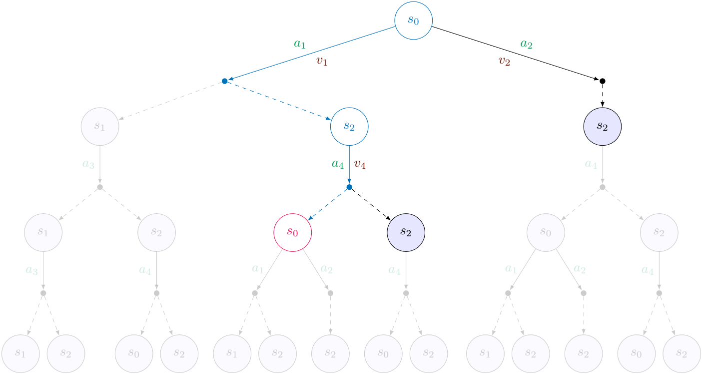
- Large unfolding $\rightsquigarrow$ heuristics
- Simulate paths to approximate values for actions
- Find the best action at the root

Monte Carlo tree search



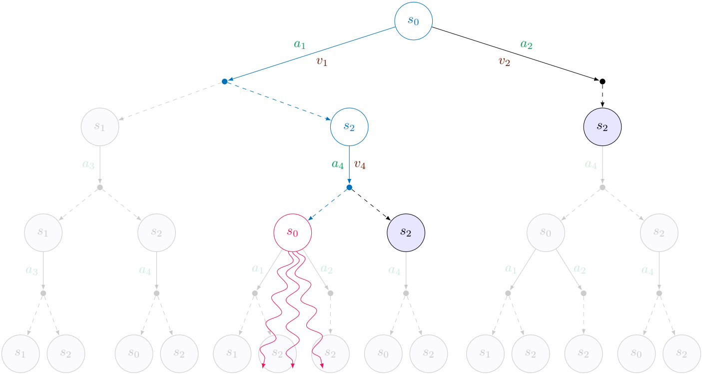
- Iterative construction of a search tree with value estimates

Monte Carlo tree search



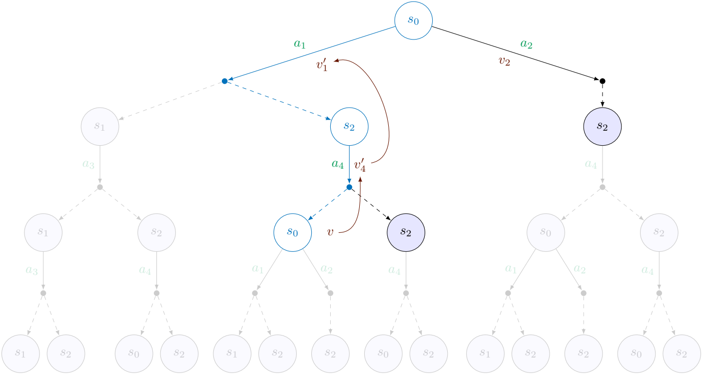
- Iterative construction of a search tree with value estimates
- Selection of a new node

Monte Carlo tree search



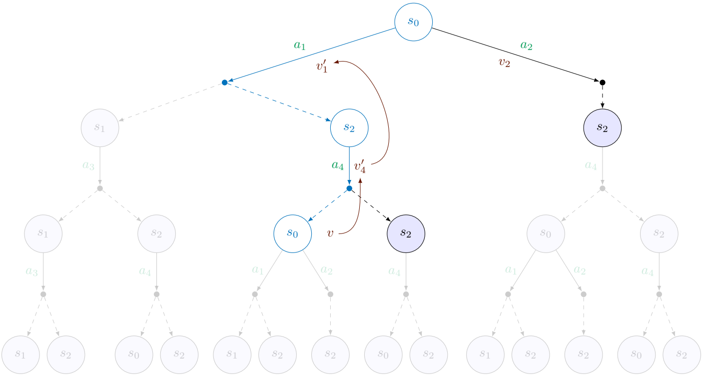
- Iterative construction of a search tree with value estimates
- Selection of a new node $\rightsquigarrow$ simulation

Monte Carlo tree search



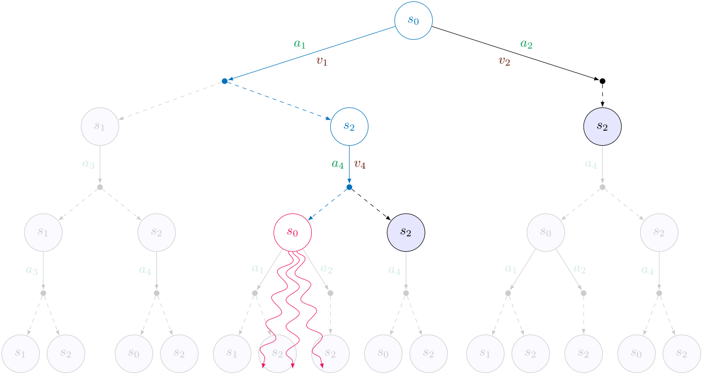
- Iterative construction of a search tree with value estimates
- Selection of a new node $\rightsquigarrow$ simulation $\rightsquigarrow$ update of the estimates

Monte Carlo tree search



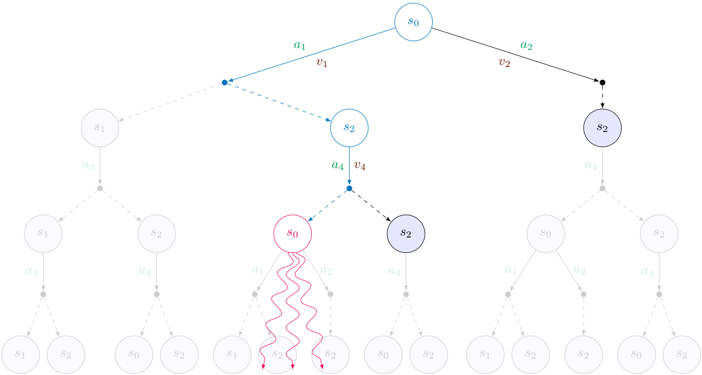
- Iterative construction of a search tree with value estimates
- Selection of a new node ⇝ simulation ⇝ update of the estimates
- The action with the best value is returned

Formal guarantees of MCTS [KS06]



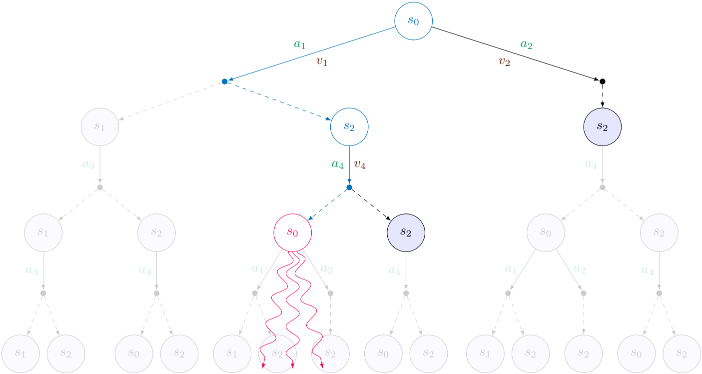
- Select using upper confidence bound for trees strategy
- After a given number of iterations n, MCTS outputs the best action

Formal guarantees of MCTS [KS06]



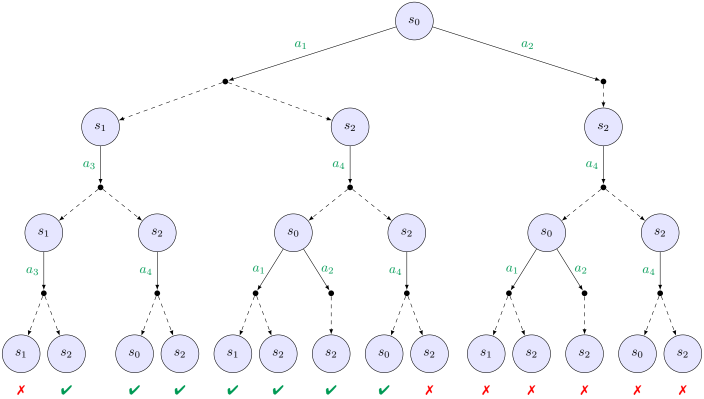
- Select using upper confidence bound for trees strategy
- After a given number of iterations n, MCTS outputs the best action
- The probability of outputting a suboptimal action converges to zero

Formal guarantees of MCTS [KS06]



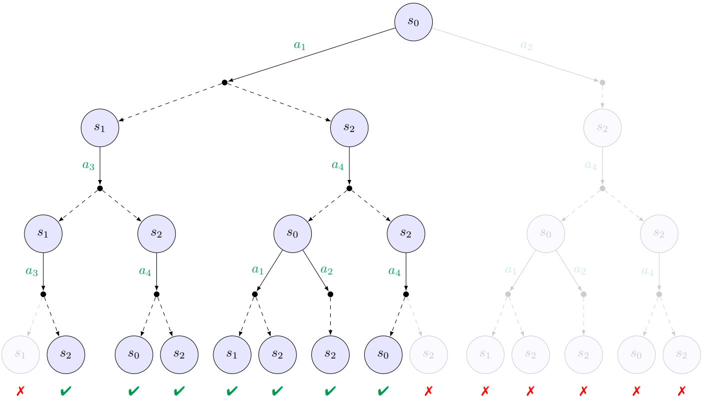
- Select using upper confidence bound for trees strategy
- After a given number of iterations n, MCTS outputs the best action
- The probability of outputting a suboptimal action converges to zero
- v_i converges to the real value of a_i at a speed of $(\log n)/n$

Advice



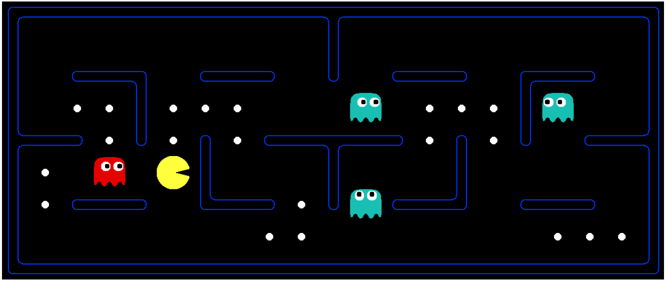
- An advice is a set of finite paths
- Defined symbolically as a logical formula φ

Advice



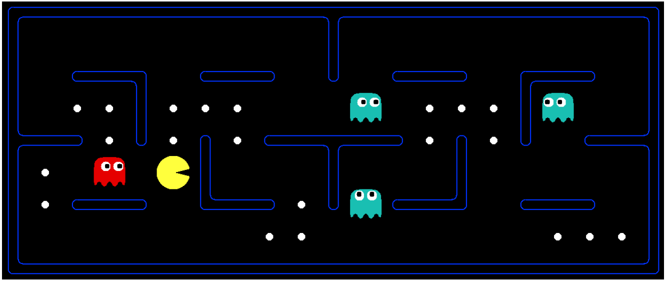
- An advice is a set of finite paths
- Defined symbolically as a logical formula φ
- φ defines a pruning of the unfolded MDP

Example of advice in PAC-MAN



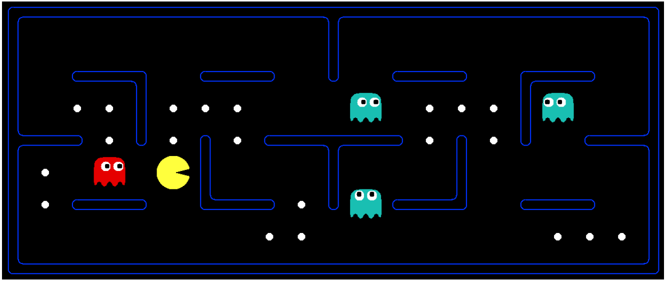
- Some states are unsafe : $\vee_g((x, y)_p = (x, y)_g)$
- Advice ψ: set of safe paths : $\square^{\leq H}\ \neg\mathrm{unsafe}$

Example of advice in PAC-MAN



- Some states are unsafe : $\vee_g((x,y)_p = (x,y)_g)$
- Advice ψ: set of safe paths : $\Box^{\leq H}\ \neg\mathrm{unsafe}$
- Stronger advice: safety is ensured no matter what stochastic transitions are taken

Example of advice in PAC-MAN



- Some states are unsafe : $\vee_g ((x,y)_p = (x,y)_g)$
- Advice ψ: set of safe paths : $\Box^{\leq H} \neg \mathrm{unsafe}$
- Stronger advice: safety is ensured no matter what stochastic transitions are taken
- Enforceable advice φ: set of paths where every action is compatible with a strategy enforcing safety

Example of advice in Pac-Man

Computation of enforceable advice φ from ψ

- A first action a_0 is compatible with φ iff

$$\forall s_1 \exists a_1 \forall s_2 \ldots, \; s_0 a_0 s_1 a_1 s_2 \ldots \models \psi$$

- Can be formulated as a 2-player game to get a strategy
- ψ can be encoded as a Boolean Formula and use QBF solver to inductively construct paths satisfying φ

Example of advice in PAC-MAN

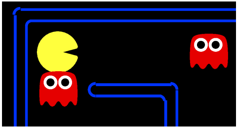
Computation of enforceable advice φ from ψ

- A first action a_0 is compatible with φ iff

$$\forall s_1 \exists a_1 \forall s_2 \ldots, \; s_0 a_0 s_1 a_1 s_2 \ldots \models \psi$$

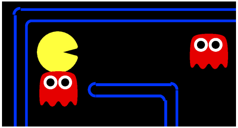
- Can be formulated as a 2-player game to get a strategy
- ψ can be encoded as a Boolean Formula and use QBF solver to inductively construct paths satisfying φ

☹ Too restrictive : there may not be a strategy $\rightsquigarrow$ Empty advice



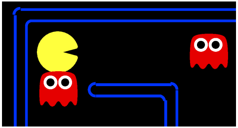
Example of advice in PAC-MAN

☹ Too restrictive : there may not be a strategy ⇝ Empty advice



Example of advice in Pac-Man

☹ Too restrictive : there may not be a strategy $\rightsquigarrow$ Empty advice



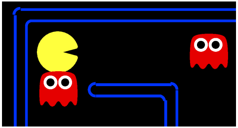
More qualitative approach

- We enforce safety as much as possible
- From state s_0, take the action maximizing probability to stay safe

$$a_0 = \arg\max_{a} \max_{\sigma \mid \sigma(s_0)=a} \mathbb{P}_\sigma(s_0 \models \psi)$$

Example of advice in PAC-MAN

☹ Too restrictive : there may not be a strategy $\rightsquigarrow$ Empty advice



More qualitative approach

- We enforce safety as much as possible
- From state s_0, take the action maximizing probability to stay safe

$$a_0 = \arg\max_a \max_{\sigma \mid \sigma(s_0)=a} \mathbb{P}_\sigma(s_0 \models \psi)$$

- This gives an advice enforced by a less restrictive strategy
- Can be computed using model checker STORM
- Calculated in a much smaller MDP (No food and faraway ghosts)

Monte Carlo tree search with advice for PAC-MAN

Monte Carlo tree search with advice for Pac-Man

- Selection advice : At the root, select among actions maximizing the probability to stay safe during the selection phase

Monte Carlo tree search with advice for Pac-Man

- Selection advice : At the root, select among actions maximizing the probability to stay safe during the selection phase
- Ensuring safety as much as possible $\rightsquigarrow$ enough to simulate among safe paths

Monte Carlo tree search with advice for Pac-Man

- Selection advice : At the root, select among actions maximizing the probability to stay safe during the selection phase
- Ensuring safety as much as possible $\rightsquigarrow$ enough to simulate among safe paths
- Simulation advice : Simulate only among safe paths

Monte Carlo tree search with advice for Pac-Man

- Selection advice : At the root, select among actions maximizing the probability to stay safe during the selection phase
- Ensuring safety as much as possible $\rightsquigarrow$ enough to simulate among safe paths
- Simulation advice : Simulate only among safe paths

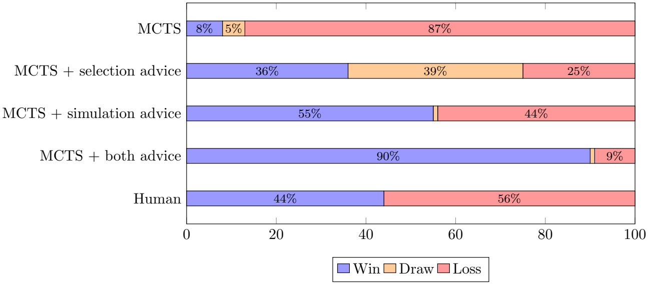
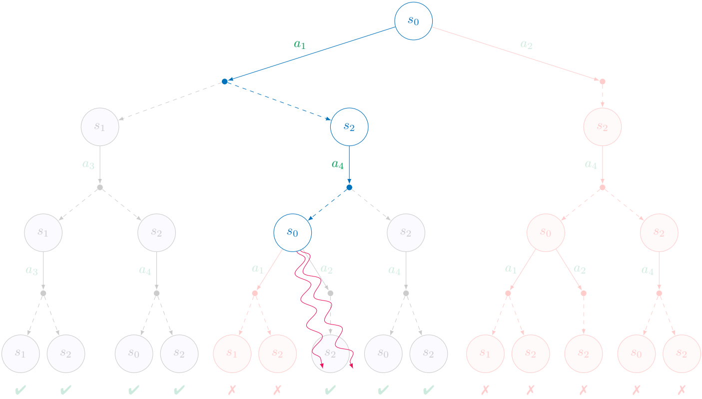
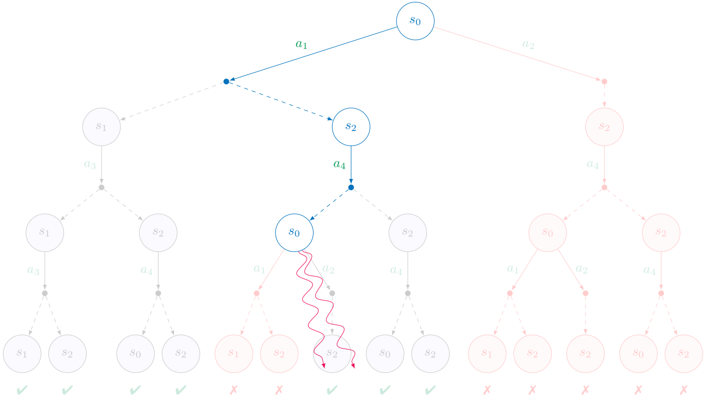


Figure: Summary of experiments for Pac-Man using MCTS with horizon 10, 40 iterations and 20 simulations per iterations. The games end in a draw after 300 steps.

Monte Carlo tree search with advice

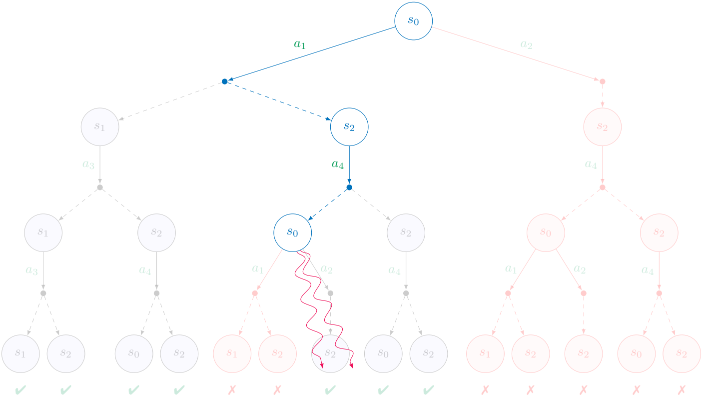


Monte Carlo tree search with advice



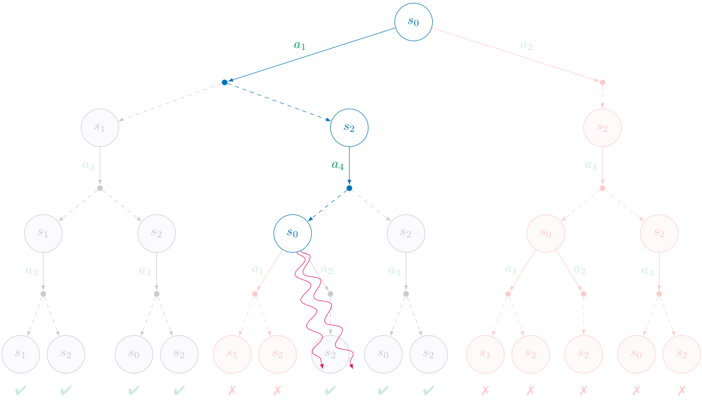
- Select actions in the unfolding pruned by a selection advice φ

Monte Carlo tree search with advice



- Select actions in the unfolding pruned by a selection advice φ
- Simulation is restricted according to a simulation advice ψ

Monte Carlo tree search with advice



- Select actions in the unfolding pruned by a selection advice φ
- Simulation is restricted according to a simulation advice ψ
- We [BCR20] show that the convergence properties are maintained:
  - for enforceable selection advice not pruning all optimal actions,
  - for all simulation advice.

Example: Scheduling of hard and soft tasks

Example: Scheduling of hard and soft tasks

Task system

- Hard tasks : should never miss deadline
- Soft tasks : positive cost for missing deadline

Example: Scheduling of hard and soft tasks

Task system

- Hard tasks : should never miss deadline
- Soft tasks : positive cost for missing deadline

Tasks are tuples (C, D, A) such that

- D relative deadline of the task,
- C : distribution over possible computation times,
- A : distribution over finitely many possible inter-arrival times.

Example: Scheduling of hard and soft tasks

Task system

- Hard tasks : should never miss deadline
- Soft tasks : positive cost for missing deadline

Tasks are tuples (C, D, A) such that

- D relative deadline of the task,
- C : distribution over possible computation times,
- A : distribution over finitely many possible inter-arrival times.

- Time is measured in CPU ticks
- Scheduler decides which active task gets CPU access
- Execution and inter-arrival time distributions are unknown to the scheduler

Task system as an MDP

States : For each tasks:

- remaining time to deadline,
- distribution over possible remaining computation times,
- distribution over possible times before next arrival.

Actions : Schedule tasks or stay idle

Stochastic transitions : Finish or kill an active task, submit a new task

Cost for soft tasks missing deadlines

Task system as an MDP

States : For each tasks:

- remaining time to deadline,
- distribution over possible remaining computation times,
- distribution over possible times before next arrival.

Actions : Schedule tasks or stay idle

Stochastic transitions : Finish or kill an active task, submit a new task

Cost for soft tasks missing deadlines

Objective : Find a strategy for scheduler that

- avoids the states where some hard task misses the deadline,
- minimizes the expected mean-cost

Task system as an MDP

- Execution and inter-arrival time distributions are not known to the scheduler
- The scheduler needs to learn the distributions using sampling before it can construct the MDP

Task system as an MDP

- Execution and inter-arrival time distributions are not known to the scheduler
- The scheduler needs to learn the distributions using sampling before it can construct the MDP

Guarantees about learning

- Probably approximately correct (PAC): for all $\epsilon, \gamma \in (0, 1)$, can approximate an ϵ-close task system, with probability $\geq 1 - \gamma$
- safely PAC learnable: PAC learnable, and can ensure safety for the hard tasks while learning
- (safely) efficiently PAC learnable : (safely) PAC learnable, and can learn in $\mathit{PTIME}\left(\text{size of the task system}, \frac{1}{\epsilon}, \frac{1}{\gamma}\right)$

Task system as an MDP

- Execution and inter-arrival time distributions are not known to the scheduler
- The scheduler needs to learn the distributions using sampling before it can construct the MDP

Guarantees about learning

- Probably approximately correct (PAC): for all $\epsilon, \gamma \in (0,1)$, can approximate an ϵ-close task system, with probability $\geq 1 - \gamma$
- safely PAC learnable: PAC learnable, and can ensure safety for the hard tasks while learning
- (safely) efficiently PAC learnable : (safely) PAC learnable, and can learn in *PTIME*$\left(\text{size of the task system}, \frac{1}{\epsilon}, \frac{1}{\gamma}\right)$

- Task systems are not always safely or efficiently PAC-learnable

Task system as an MDP

- Execution and inter-arrival time distributions are not known to the scheduler
- The scheduler needs to learn the distributions using sampling before it can construct the MDP

Guarantees about learning

- Probably approximately correct (PAC): for all $\epsilon, \gamma \in (0, 1)$, can approximate an ϵ-close task system, with probability $\geq 1 - \gamma$
- safely PAC learnable: PAC learnable, and can ensure safety for the hard tasks while learning
- (safely) efficiently PAC learnable : (safely) PAC learnable, and can learn in *PTIME*$\left(\text{size of the task system}, \frac{1}{\epsilon}, \frac{1}{\gamma}\right)$

- Task systems are not always safely or efficiently PAC-learnable
- Sufficient conditions for safe and efficient PAC-learning [BCG+21]

Scheduling of hard and soft tasks

- Model checkers can handle only relatively small task systems
- Use learning-based methods : MCTS, deep Q-learning
- Advice enforceable by safe strategies in MCTS
- Safe strategies to shield simulations during deep Q-learning

Scheduling of hard and soft tasks

- Model checkers can handle only relatively small task systems
- Use learning-based methods : MCTS, deep Q-learning
- Advice enforceable by safe strategies in MCTS
- Safe strategies to shield simulations during deep Q-learning

Earliest deadline first strategy

- Schedule hard task with earliest deadline
- If no hard tasks are active, schedule soft tasks

☺ Strategy ensuring safety ☺ Easy to calculate ☹ Too restrictive

Scheduling of hard and soft tasks

- Model checkers can handle only relatively small task systems
- Use learning-based methods : MCTS, deep Q-learning
- Advice enforceable by safe strategies in MCTS
- Safe strategies to shield simulations during deep Q-learning

Earliest deadline first strategy

- Schedule hard task with earliest deadline
- If no hard tasks are active, schedule soft tasks

☺ Strategy ensuring safety ☺ Easy to calculate ☹ Too restrictive

Most general safe scheduler

- Allow all actions compatible with any safe strategy

☺ Allows maximal exploration ☹ Needs to be precomputed

Scheduling of hard and soft tasks

Average cost over 600 steps for task systems

Task	MDP size	STORM output	MCTS unsafe	MCTS EDF	MCTS MGS	DQL unsafe	DQL EDF	DQL MGS
4S	10^5	0.38	0.52	N/A	N/A	0.56	N/A	N/A
5S	10^6	TimeOut	0	N/A	N/A	0.13	N/A	N/A
10S	10^{18}	TimeOut	0	N/A	N/A	0.96	N/A	N/A
1H, 2S	10^4	0.07	0.67	0.28	0.14	0.24	0.22	0.11
1H, 3S	10^5	0.28	1.13	0.49	0.45	∞	0.47	0.47
2H, 1S	10^4	0	0.92	0.2	0	∞	0.3	0.02
2H, 5S	10^{10}	TimeOut	3.44	2.14	1.93	∞	2.48	2.39
3H, 6S	10^{14}	TimeOut	4.17	2.97	2.88	∞	3.47	3.42
2H, 10S	10^{22}	TimeOut	0.3	0.03	0.03	∞	1.6	1.42
4H, 12S	10^{30}	TimeOut	2.1	1.3	1.2	∞	2.87	2.68

Comparison of MCTS and deep Q-learning.

Scheduling of hard and soft tasks

Average cost over 600 steps for task systems

Task	MDP size	STORM output	MCTS unsafe	MCTS EDF	MCTS MGS	DQL unsafe	DQL EDF	DQL MGS
4S	10^5	0.38	0.52	N/A	N/A	0.56	N/A	N/A
5S	10^6	TimeOut	0	N/A	N/A	0.13	N/A	N/A
10S	10^{18}	TimeOut	0	N/A	N/A	0.96	N/A	N/A
1H, 2S	10^4	0.07	0.67	0.28	0.14	0.24	0.22	0.11
1H, 3S	10^5	0.28	1.13	0.49	0.45	∞	0.47	0.47
2H, 1S	10^4	0	0.92	0.2	0	∞	0.3	0.02
2H, 5S	10^{10}	TimeOut	3.44	2.14	1.93	∞	2.48	2.39
3H, 6S	10^{14}	TimeOut	4.17	2.97	2.88	∞	3.47	3.42
2H, 10S	10^{22}	TimeOut	0.3	0.03	0.03	∞	1.6	1.42
4H, 12S	10^{30}	TimeOut	2.1	1.3	1.2	∞	2.87	2.68

Comparison of MCTS and deep Q-learning.

Scheduling of hard and soft tasks

Average cost over 600 steps for task systems

Task	MDP size	STORM output	MCTS unsafe	MCTS EDF	MCTS MGS	DQL unsafe	DQL EDF	DQL MGS
4S	10^5	0.38	0.52	N/A	N/A	0.56	N/A	N/A
5S	10^6	TimeOut	0	N/A	N/A	0.13	N/A	N/A
10S	10^{18}	TimeOut	0	N/A	N/A	0.96	N/A	N/A
1H, 2S	10^4	0.07	0.67	0.28	0.14	0.24	0.22	0.11
1H, 3S	10^5	0.28	1.13	0.49	0.45	∞	0.47	0.47
2H, 1S	10^4	0	0.92	0.2	0	∞	0.3	0.02
2H, 5S	10^{10}	TimeOut	3.44	2.14	1.93	∞	2.48	2.39
3H, 6S	10^{14}	TimeOut	4.17	2.97	2.88	∞	3.47	3.42
2H, 10S	10^{22}	TimeOut	0.3	0.03	0.03	∞	1.6	1.42
4H, 12S	10^{30}	TimeOut	2.1	1.3	1.2	∞	2.87	2.68

Comparison of MCTS and deep Q-learning.

Scheduling of hard and soft tasks

Average cost over 600 steps for task systems

Task	MDP size	STORM output	MCTS unsafe	MCTS EDF	MCTS MGS	DQL unsafe	DQL EDF	DQL MGS
4S	10^5	0.38	0.52	N/A	N/A	0.56	N/A	N/A
5S	10^6	TimeOut	0	N/A	N/A	0.13	N/A	N/A
10S	10^{18}	TimeOut	0	N/A	N/A	0.96	N/A	N/A
1H, 2S	10^4	0.07	0.67	0.28	0.14	0.24	0.22	0.11
1H, 3S	10^5	0.28	1.13	0.49	0.45	∞	0.47	0.47
2H, 1S	10^4	0	0.92	0.2	0	∞	0.3	0.02
2H, 5S	10^{10}	TimeOut	3.44	2.14	1.93	∞	2.48	2.39
3H, 6S	10^{14}	TimeOut	4.17	2.97	2.88	∞	3.47	3.42
2H, 10S	10^{22}	TimeOut	0.3	0.03	0.03	∞	1.6	1.42
4H, 12S	10^{30}	TimeOut	2.1	1.3	1.2	∞	2.87	2.68

Comparison of MCTS and deep Q-learning.

Scheduling of hard and soft tasks

Average cost over 600 steps for task systems

Task	MDP size	Storm output	MCTS unsafe	MCTS EDF	MCTS MGS	DQL unsafe	DQL EDF	DQL MGS
4S	10^5	0.38	0.52	N/A	N/A	0.56	N/A	N/A
5S	10^6	TimeOut	0	N/A	N/A	0.13	N/A	N/A
10S	10^{18}	TimeOut	0	N/A	N/A	0.96	N/A	N/A
1H, 2S	10^4	0.07	0.67	0.28	0.14	0.24	0.22	0.11
1H, 3S	10^5	0.28	1.13	0.49	0.45	∞	0.47	0.47
2H, 1S	10^4	0	0.92	0.2	0	∞	0.3	0.02
2H, 5S	10^{10}	TimeOut	3.44	2.14	1.93	∞	2.48	2.39
3H, 6S	10^{14}	TimeOut	4.17	2.97	2.88	∞	3.47	3.42
2H, 10S	10^{22}	TimeOut	0.3	0.03	0.03	∞	1.6	1.42
4H, 12S	10^{30}	TimeOut	2.1	1.3	1.2	∞	2.87	2.68

Comparison of MCTS and deep Q-learning.

Scheduling of hard and soft tasks

Average cost over 600 steps for task systems

Task	MDP size	STORM output	MCTS unsafe	MCTS EDF	MCTS MGS	DQL unsafe	DQL EDF	DQL MGS
4S	10^5	0.38	0.52	N/A	N/A	0.56	N/A	N/A
5S	10^6	TimeOut	0	N/A	N/A	0.13	N/A	N/A
10S	10^{18}	TimeOut	0	N/A	N/A	0.96	N/A	N/A
1H, 2S	10^4	0.07	0.67	0.28	0.14	0.24	0.22	0.11
1H, 3S	10^5	0.28	1.13	0.49	0.45	∞	0.47	0.47
2H, 1S	10^4	0	0.92	0.2	0	∞	0.3	0.02
2H, 5S	10^{10}	TimeOut	3.44	2.14	1.93	∞	2.48	2.39
3H, 6S	10^{14}	TimeOut	4.17	2.97	2.88	∞	3.47	3.42
2H, 10S	10^{22}	TimeOut	0.3	0.03	0.03	∞	1.6	1.42
4H, 12S	10^{30}	TimeOut	2.1	1.3	1.2	∞	2.87	2.68

Comparison of MCTS and deep Q-learning.

Neural advice

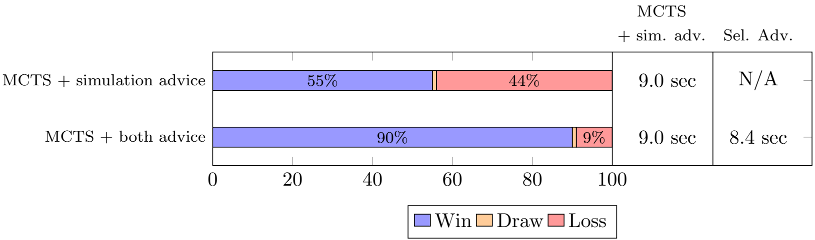


Figure : Summary of experiments for PAC-MAN using MCTS with a simulation advice.

Neural advice

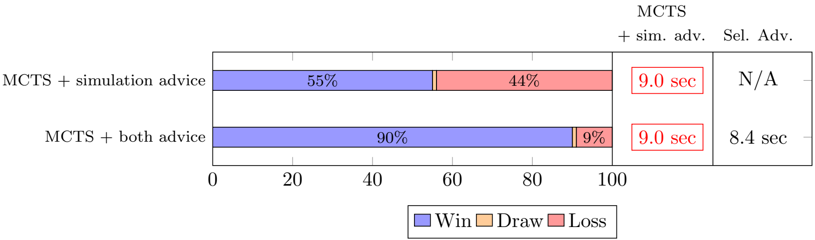


Figure : Summary of experiments for PAC-MAN using MCTS with a simulation advice.

Neural advice

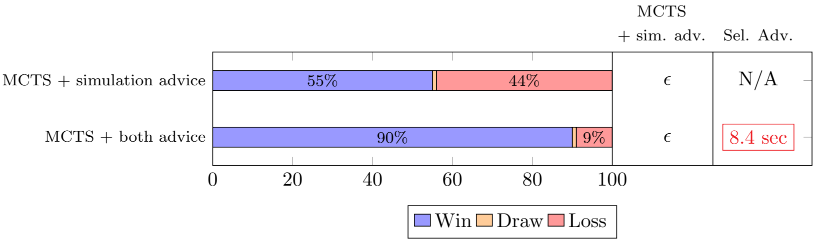


Figure : Summary of experiments for Pac-Man using MCTS with a simulation advice.

- Using model checker for advice is costly in terms of time

Neural advice

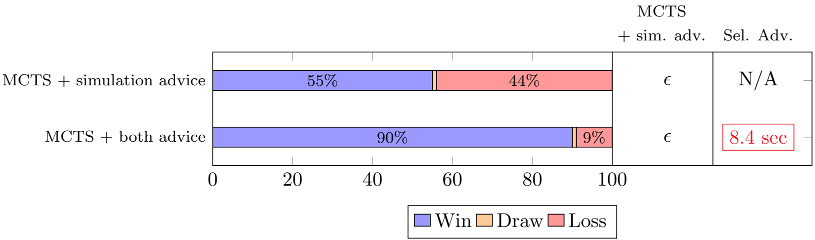


Figure : Summary of experiments for PAC-MAN using MCTS with a simulation advice.

- Using model checker for advice is costly in terms of time
- Neural advice : Use a neural network to imitate an advice

Neural advice

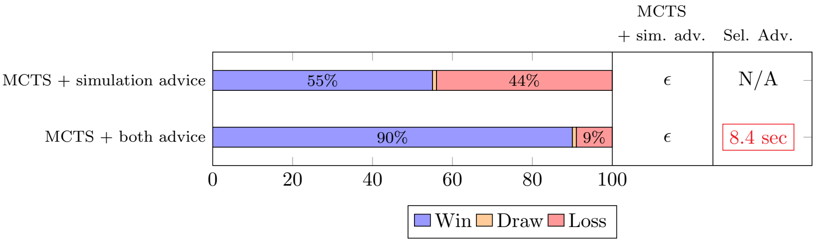
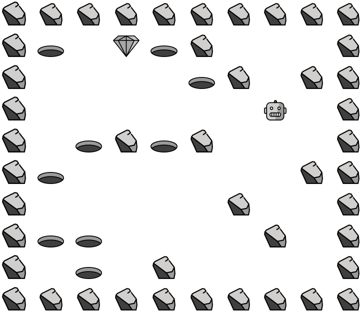


Figure : Summary of experiments for PAC-MAN using MCTS with a simulation advice.

- Using model checker for advice is costly in terms of time
- Neural advice : Use a neural network to imitate an advice
- Imitation learning : Framework to mimic a strategy

Example : Frozen Lake



- Controller: Robot
- Slips to different direction with small probability
- Reward for reaching target

- States: position of the robot
- Actions: Robot's decision
- Stochastic transitions: Robot's actual move

Example : Frozen Lake

Exact algorithm

- Strategy maximizing probability to reach the target quickly

$$\mathrm{Opt}(s) = \arg\max_{a} \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_{\sigma}(s \models \Diamond \text{ target})$$

$$f_{\sigma}(s,a) = \begin{cases} \min_{\sigma \mid \sigma(s)=a} \mathbb{E}(\text{distance to target}) & \text{if } a \in \mathrm{Opt}(s) \\ \infty & \text{otherwise.} \end{cases}$$

- $\sigma(s) = \arg\min_a (f_{\sigma}(s,a))$

Example : Frozen Lake

Exact algorithm

- Strategy maximizing probability to reach the target quickly

$$\mathrm{Opt}(s) = \arg\max_{a} \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_\sigma(s \models \Diamond \text{ target})$$

$$f_\sigma(s,a) = \begin{cases} \min_{\sigma \mid \sigma(s)=a} \mathbb{E}(\text{distance to target}) & \text{if } a \in \mathrm{Opt}(s) \\ \infty & \text{otherwise.} \end{cases}$$

- $\sigma(s) = \arg\min_a(f_\sigma(s,a))$

Heuristic algorithm

- Monte Carlo tree search : $\sigma(s) = \arg\max_a \text{value}(s,a)$

Example : Frozen Lake

Exact algorithm

- Strategy maximizing probability to reach the target quickly

$$\mathrm{Opt}(s) = \arg\max_a \max_{\sigma|\sigma(s)=a} \mathbb{P}_\sigma(s \models \Diamond \text{ target})$$

$$f_\sigma(s,a) = \begin{cases} \min_{\sigma|\sigma(s)=a} \mathbb{E}(\text{distance to target}) & \text{if } a \in \mathrm{Opt}(s) \\ \infty & \text{otherwise.} \end{cases}$$

- $\sigma(s) = \arg\min_a(f_\sigma(s,a))$

Heuristic algorithm

- Monte Carlo tree search : $\sigma(s) = \arg\max_a \text{value}(s,a)$

Imitation learning

- Train a neural network NN to learn f_σ or value

Example : Frozen Lake

Exact algorithm

- Strategy maximizing probability to reach the target quickly

$$\mathrm{Opt}(s) = \arg\max_{a} \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_{\sigma}(s \models \Diamond \text{ target})$$

$$f_{\sigma}(s,a) = \begin{cases} \min_{\sigma \mid \sigma(s)=a} \mathbb{E}(\text{distance to target}) & \text{if } a \in \mathrm{Opt}(s) \\ \infty & \text{otherwise.} \end{cases}$$

- $\sigma(s) = \arg\min_a (f_{\sigma}(s,a))$

Heuristic algorithm

- Monte Carlo tree search : $\sigma(s) = \arg\max_a \mathrm{value}(s,a)$

Imitation learning

- Train a neural network NN to learn f_{σ} or value
- Learnt strategy : $\sigma_{\text{learnt}}(s) = \arg\mathrm{opt}_a \, \mathrm{NN}(s,a)$

Example : Frozen Lake

Evaluating the learnt strategy

Example : Frozen Lake

Evaluating the learnt strategy

- Traditional approaches : loss function, accuracy

Example : Frozen Lake

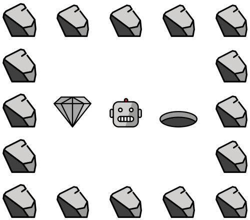
Evaluating the learnt strategy

- Traditional approaches : loss function, accuracy
- Traditional approaches may not be sufficient

Example : Frozen Lake

Evaluating the learnt strategy

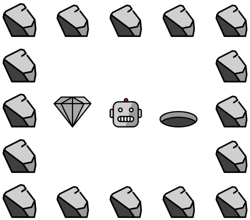
- Traditional approaches : loss function, accuracy
- Traditional approaches may not be sufficient
  - Error in one critical state can be enough for the stategy to perform bad



Example : Frozen Lake

Evaluating the learnt strategy

- Traditional approaches : loss function, accuracy
- Traditional approaches may not be sufficient
  - Error in one critical state can be enough for the stategy to perform bad



- Use statistical model checking to compare the strategies
  - Simulate a set of paths and compare statistics

Example : Frozen Lake

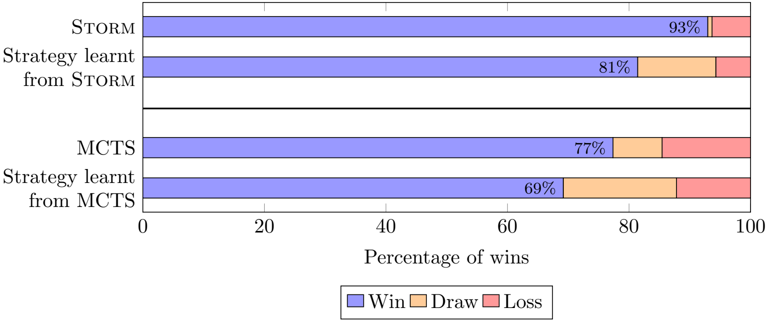


Figure: Imitation learning of perfect vs MCTS-based strategies

Example : Frozen Lake

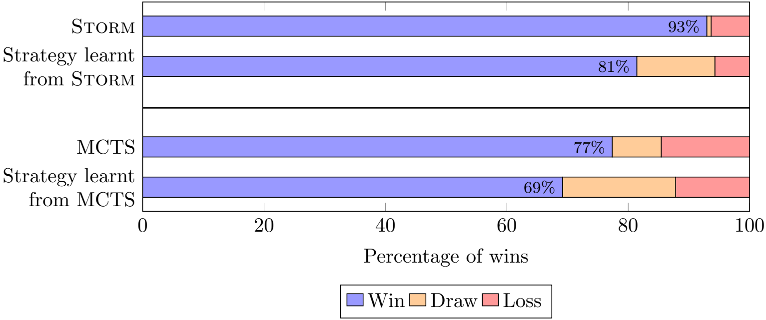


Figure: Imitation learning of perfect vs MCTS-based strategies

- Data from exact methods $\rightsquigarrow$ noise-free data $\rightsquigarrow$ better learnt strategy

Neural advice in Pac-Man

Neural advice in Pac-Man

- Symbolic advice enforced by the strategy which takes the action maximizing probability to stay safe

$$\sigma(s) = \arg\max_{a} \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_{\sigma}(s \models \Box^{\leq H} \neg \mathrm{unsafe})$$

Neural advice in Pac-Man

- Symbolic advice enforced by the strategy which takes the action maximizing probability to stay safe

$$\sigma(s) = \arg\max_{a} \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_\sigma(s \models \Box^{\leq H} \neg \mathrm{unsafe})$$

- Too costly to compute the advice in terms of time

Neural advice in Pac-Man

- Symbolic advice enforced by the strategy which takes the action maximizing probability to stay safe

$$\sigma(s) = \arg\max_{a} \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_{\sigma}(s \models \Box^{\leq H} \neg \mathrm{unsafe})$$

- Too costly to compute the advice in terms of time
- Train a neural network offline to learn

$$f_{\sigma}(s, a) = \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_{\sigma}(s \models \Box^{\leq H} \neg \mathrm{unsafe})$$

Neural advice in Pac-Man

- Symbolic advice enforced by the strategy which takes the action maximizing probability to stay safe

$$\sigma(s) = \arg\max_a \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_\sigma(s \models \square^{\leq H} \neg \mathrm{unsafe})$$

- Too costly to compute the advice in terms of time
- Train a neural network offline to learn

$$f_\sigma(s, a) = \max_{\sigma \mid \sigma(s)=a} \mathbb{P}_\sigma(s \models \square^{\leq H} \neg \mathrm{unsafe})$$

- Neural advice enforced by the strategy which takes the action maximizing value given by the network

$$\sigma_{\mathrm{NN}}(s) = \arg\max_a \mathrm{NN}(s, a)$$

Neural advice in PAC-MAN

How we should generate data for the neural network?

- Randomly generate states and actions

☹ Poor performance

Neural advice in PAC-MAN

How we should generate data for the neural network?

- Randomly generate states and actions

☹ Poor performance

Dataset aggregation algorithm (DAgger)

Iteratively add more data in the training dataset

- Generate states from running simulations using learnt strategy
- Add new states to the dataset and learn a new strategy

☺ Realistic view of the states frequently encountered

Neural advice in Pac-Man

How we should generate data for the neural network?

- Randomly generate states and actions

☹ Poor performance

Dataset aggregation algorithm (DAgger)

Iteratively add more data in the training dataset

- Generate states from running simulations using learnt strategy
- Add new states to the dataset and learn a new strategy

☺ Realistic view of the states frequently encountered

Sharp dataset aggregation algorithm (Sharp DAgger)

- Add counter-example to the dataset if the neural network is performing poorly

☺ Focuses at finding states where correct decision is crucial

Neural advice in Pac-Man

Learning a neural advice

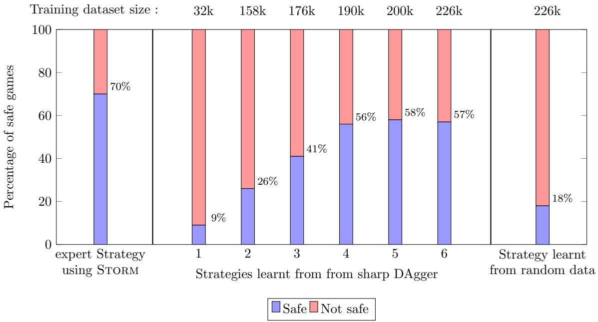


Figure: Sharp DAgger for strategy to stay safe in Pac-Man

Neural advice in PAC-MAN

Learning a neural advice

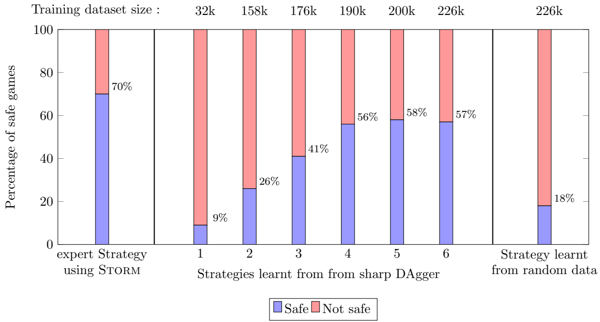


Figure: Sharp DAgger for strategy to stay safe in PAC-MAN

- Symbolic advice enforceable by a strategy with 70% safety rate
- Neural advice enforceable by a strategy with 58% safety rate
- Strategy learnt from random data has 18% safety rate

Experimental results : PAC-MAN

Monte Carlo tree search with neural advice

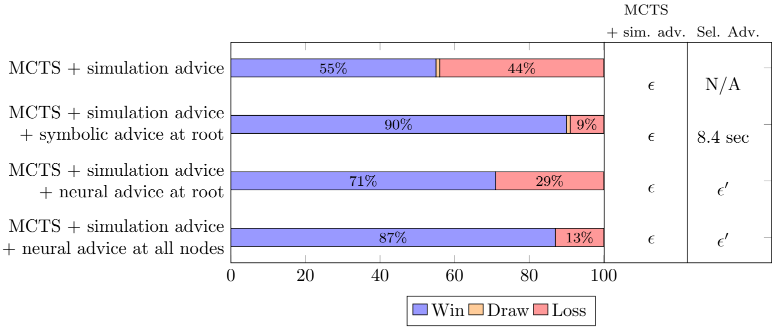


Figure: Summary of experiments with neural advice for PAC-MAN

Conclusion

Conclusion

- How to inject domain knowledge in MCTS?
  - Symbolic advice for selection and simulation

Conclusion

- How to inject domain knowledge in MCTS?
  - Symbolic advice for selection and simulation
- How to preserve the convergence guarantees of MCTS?
  - Enforceable advice with an optimality assumption

Conclusion

- How to inject domain knowledge in MCTS?
  - Symbolic advice for selection and simulation
- How to preserve the convergence guarantees of MCTS?
  - Enforceable advice with an optimality assumption
- How to implement them?
  - Using model checkers

Conclusion

- How to inject domain knowledge in MCTS?
  - Symbolic advice for selection and simulation
- How to preserve the convergence guarantees of MCTS?
  - Enforceable advice with an optimality assumption
- How to implement them?
  - Using model checkers
- How to decrease computation time for implementing symbolic advice?
  - By imitating the advice by neural network

Conclusion

- How to inject domain knowledge in MCTS?
  - Symbolic advice for selection and simulation
- How to preserve the convergence guarantees of MCTS?
  - Enforceable advice with an optimality assumption
- How to implement them?
  - Using model checkers
- How to decrease computation time for implementing symbolic advice?
  - By imitating the advice by neural network
- How to generate data for the neural network?
  - Using a counter-example guided dataset aggregation loop

Conclusion

- How to inject domain knowledge in MCTS?
  - Symbolic advice for selection and simulation
- How to preserve the convergence guarantees of MCTS?
  - Enforceable advice with an optimality assumption
- How to implement them?
  - Using model checkers
- How to decrease computation time for implementing symbolic advice?
  - By imitating the advice by neural network
- How to generate data for the neural network?
  - Using a counter-example guided dataset aggregation loop
- Does it work?
  - Good results with multiple examples

Thank You!