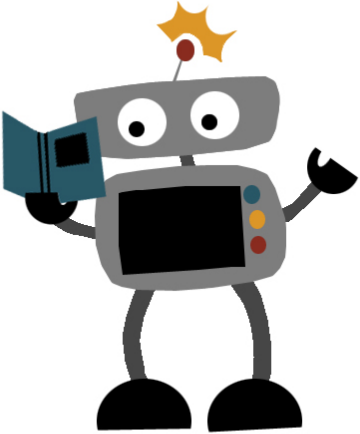
Sample-efficient Monte-Carlo planning

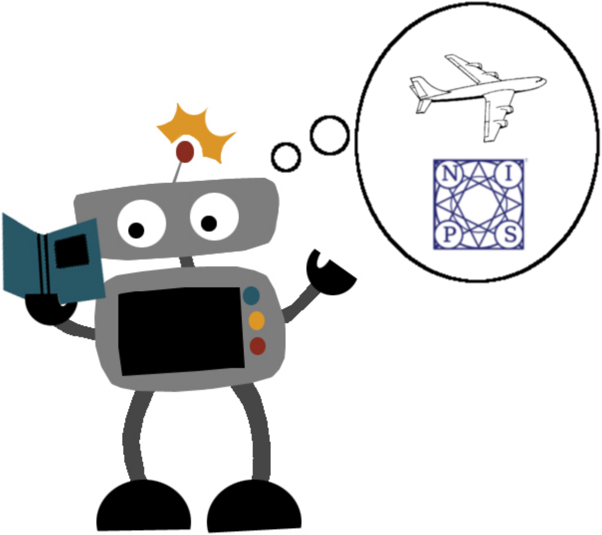
J. Grill, M. Valko, R. Munos

December 6, 2016

Planning in Markov Decision Processes



Planning in Markov Decision Processes



Planning in Markov Decision Processes

- We can perform actions to impact the environment.

Planning in Markov Decision Processes

- We can perform actions to impact the environment.
- We receive a reward and an observation of the environment change.

Planning in Markov Decision Processes

- We can perform actions to impact the environment.
- We receive a reward and an observation of the environment change.
- The environment modifications and the rewards are stochastic.

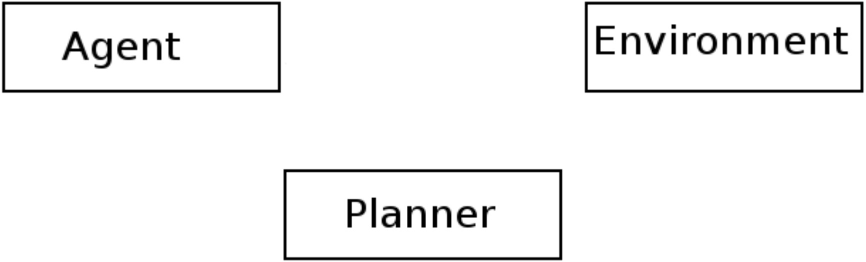
Planning in Markov Decision Processes

- We can perform actions to impact the environment.
- We receive a reward and an observation of the environment change.
- The environment modifications and the rewards are stochastic.
- We have a generative model.

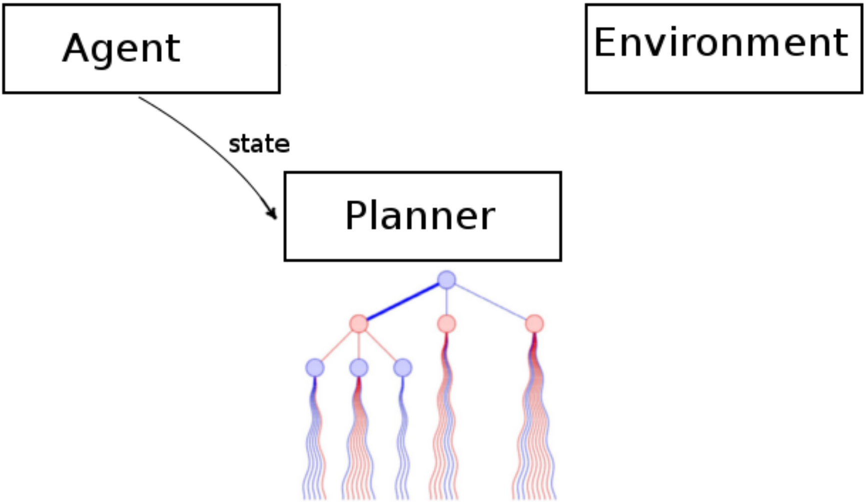
Planning in Markov Decision Processes

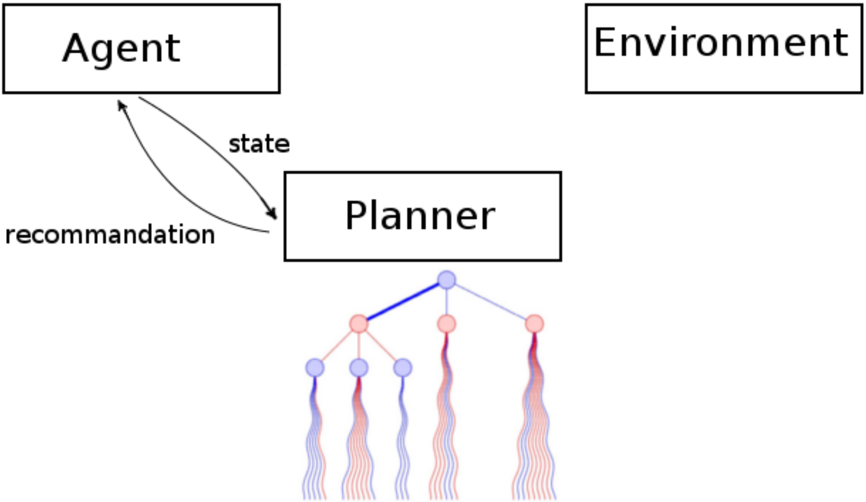
- We can perform actions to impact the environment.
- We receive a reward and an observation of the environment change.
- The environment modifications and the rewards are stochastic.
- We have a generative model.
- We are only interested in the policy for our current environment configuration.

Planning in Markov Decision Processes

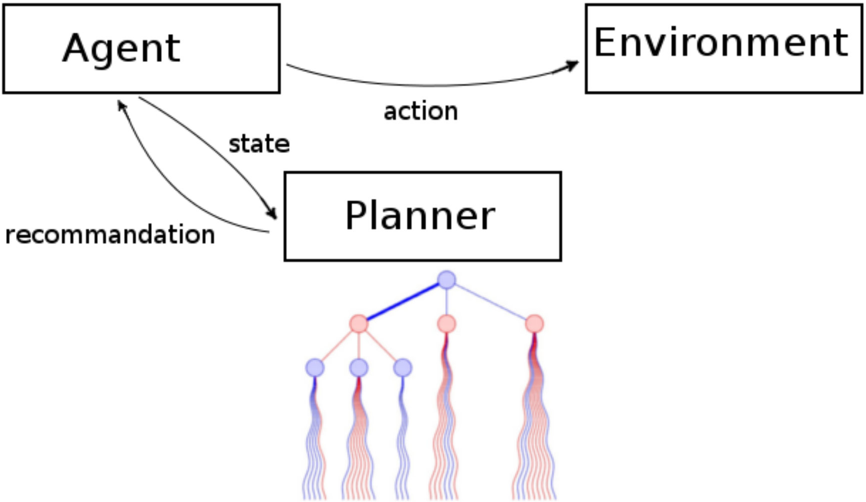


Planning in Markov Decision Processes

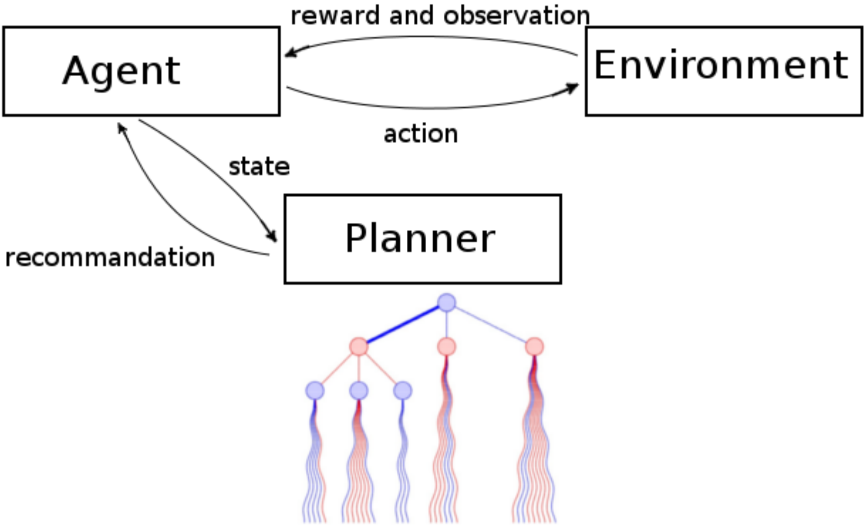

Agent
Environment
state
Planner



Agent
Environment
state
Planner
recommandation

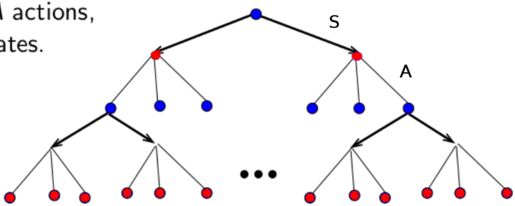

Agent
Environment
action
state
Planner
recommandation

Planning in Markov Decision Processes



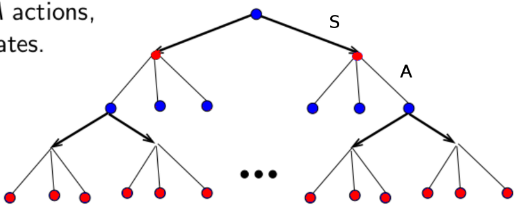
Tree representation

Assume A actions,
S next-states.



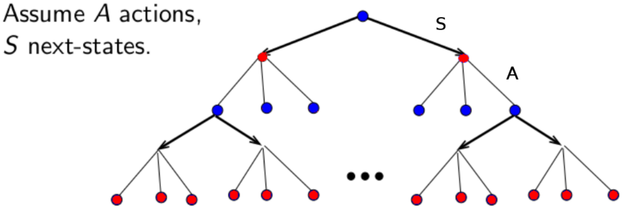
Tree representation

Assume A actions,
S next-states.



A node s: Description of the environment
$\mathcal{V}[s]$: discounted sum of rewards you get if you play optimally from s.

Tree representation

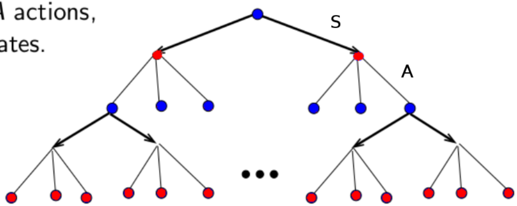


A node s: Description of the environment
$\mathcal{V}[s]$: discounted sum of rewards you get if you play optimally from s.

- Maximum nodes (agent): $$\mathcal{V}[s] = \max_{s' \text{ child of } s} \mathcal{V}[s'].$$

Tree representation

Assume A actions,
S next-states.

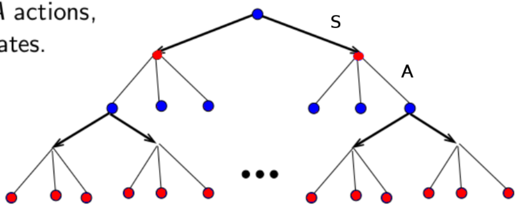


A node s: Description of the environment
$\mathcal{V}[s]$: discounted sum of rewards you get if you play optimally from s.

- Maximum nodes (agent): $\mathcal{V}[s] = \max_{s' \text{ child of } s} \mathcal{V}[s'].$
- Average nodes (environment): $\mathcal{V}[s] = r(s) + \gamma \sum_{s' \text{ child of } s} p(s'|s)\mathcal{V}[s'].$

Tree representation

Assume A actions,
S next-states.



A node s: Description of the environment
$\mathcal{V}[s]$: discounted sum of rewards you get if you play optimally from s.

- Maximum nodes (agent): $\mathcal{V}[s] = \max_{s' \text{ child of } s} \mathcal{V}[s']$.
- Average nodes (environment): $\mathcal{V}[s] = r(s) + \gamma \sum_{s' \text{ child of } s} p(s'|s)\mathcal{V}[s']$.

Goal: Compute the value of the root $\mathcal{V}[s_0]$.

Hypothesis

We assume the access to a **generative model:**

$$\text{average node } s \longrightarrow \begin{array}{ll} r_s & \text{reward sample s.t. } \mathbb{E}r = r(s) \\ y_s & \text{next state sample} \sim p(\cdot|s) \end{array}$$

Hypothesis

We assume the access to a **generative model:**

$$\text{average node } s \longrightarrow \begin{array}{ll} r_s & \text{reward sample s.t. } \mathbb{E}r = r(s) \\ y_s & \text{next state sample} \sim p(\cdot|s) \end{array}$$

We do not assume to know the transition or reward probability law.

Hypothesis

We assume the access to a **generative model:**

$$\text{average node } s \longrightarrow \begin{array}{ll} r_s & \text{reward sample s.t. } \mathbb{E}r = r(s) \\ y_s & \text{next state sample} \sim p(\cdot|s) \end{array}$$

We do not assume to know the transition or reward probability law.

PAC (Probably Approximately Correct)

For any $\delta > 0, \epsilon > 0$, we compute $v(\delta, \epsilon)$ such that

$$\mathbb{P}\left[|v(\delta,\epsilon) - \mathcal{V}[s_0]| < \epsilon\right] > 1 - \delta$$

Adaptive Algorithm

Sample complexity: the number of calls to the generative model.

Sample complexity: the number of calls to the generative model.

The number of nodes of depth h: $(AS)^h$.

Adaptive Algorithm

Sample complexity: the number of calls to the generative model.

The number of nodes of depth h: $(AS)^h$.

Design **adaptive** algorithms that doesn't explore uniformly the whole tree.

The Algorithm: `TrailBlazer`

Each node is a persistent object with its own memory.

The Algorithm: `TrailBlazer`

Each node is a persistent object with its own memory.

It can be queried for an estimation of its node.

The Algorithm: `TrailBlazer`

Each node is a persistent object with its own memory.

It can be queried for an estimation of its node.

To compute this estimation it can:

- Perform call to the generative model.

The Algorithm: `TrailBlazer`

Each node is a persistent object with its own memory.

It can be queried for an estimation of its node.

To compute this estimation it can:

- Perform call to the generative model.
- Query its children for their value.

The Algorithm: `TrailBlazer`

Each node is a persistent object with its own memory.

It can be queried for an estimation of its node.

To compute this estimation it can:

- Perform call to the generative model.
- Query its children for their value.

Queries are performed with a precision as argument: ϵ and n.

The Algorithm: `TrailBlazer`

Each node is a persistent object with its own memory.

It can be queried for an estimation of its node.

To compute this estimation it can:

- Perform call to the generative model.
- Query its children for their value.

Queries are performed with a precision as argument: ϵ and n.

- The bias of the estimator is of order ϵ
- The variance of the estimator of order $1/n$.

The Algorithm: `TrailBlazer`

Average node:

- Sample n transitions and n rewards.
- Query the sampled children with bias ϵ/γ.

The Algorithm: `TrailBlazer`

Average node:

- Sample n transitions and n rewards.
- Query the sampled children with bias ϵ/γ.

Maximum node:

- Run best arm identification sub-routine.
- Query the best arm with a high a variance query.

The Algorithm: `TrailBlazer`

Average node:

- Sample n transitions and n rewards.
- Query the sampled children with bias ϵ/γ.

Maximum node:

- Run best arm identification sub-routine.
- Query the best arm with a high a variance query.

The algorithm `TrailBlazer`

- behaves like Monte-Carlo sampling when there are no max node.

The Algorithm: `TrailBlazer`

Average node:

- Sample n transitions and n rewards.
- Query the sampled children with bias ϵ/γ.

Maximum node:

- Run best arm identification sub-routine.
- Query the best arm with a high a variance query.

The algorithm `TrailBlazer`

- behaves like Monte-Carlo sampling when there are no max node.
- is computationally efficient and easy to implement.

The Algorithm: `TrailBlazer`

Average node:

- Sample n transitions and n rewards.
- Query the sampled children with bias ϵ/γ.

Maximum node:

- Run best arm identification sub-routine.
- Query the best arm with a high a variance query.

The algorithm `TrailBlazer`

- behaves like Monte-Carlo sampling when there are no max node.
- is computationally efficient and easy to implement.
- is adaptive.

Sample complexity in the finite S case

UCT: [L. Kocsis and C. Szepesvári, 2006]

Asymptotic analysis but no finite time guarantees.

UCT: [L. Kocsis and C. Szepesvári, 2006]

Asymptotic analysis but no finite time guarantees.

StoP: [B. Szörényi et al, 2014]

Explore $(\kappa S)^h$ nodes instead of $(AS)^h$.

Sample complexity in the finite S case

UCT: [L. Kocsis and C. Szepesvári, 2006]

Asymptotic analysis but no finite time guarantees.

StoP: [B. Szörényi et al, 2014]

Explore $(\kappa S)^h$ nodes instead of $(AS)^h$.

The quantity $\kappa \in [1, A]$ is problem dependent.
It measures the branching factor of the set of "important" states.

Sample complexity in the finite S case

UCT: [L. Kocsis and C. Szepesvári, 2006]

Asymptotic analysis but no finite time guarantees.

StoP: [B. Szörényi et al, 2014]

Explore $(\kappa S)^h$ nodes instead of $(AS)^h$.

The quantity $\kappa \in [1, A]$ is problem dependent.
It measures the branching factor of the set of "important" states.

Problem: algorithm consider the set of all policies which grows exponentially with the number of states.

Sample complexity in the finite S case

Sample complexity bound of StoP:

$$\mathcal{O}\left((1/\epsilon)^{2+\frac{\log(\kappa S)}{\log(1/\gamma)}}\right)$$

Sample complexity in the finite S case

Sample complexity bound of StoP:

$$\mathcal{O}\left((1/\epsilon)^{2+\frac{\log(\kappa S)}{\log(1/\gamma)}}\right)$$

Complexity of uniform planning:

$$\mathcal{O}\left((1/\epsilon)^{2+\frac{\log(AS)}{\log(1/\gamma)}}\right)$$

Sample complexity in the finite S case

Sample complexity bound of StoP:

$$\mathcal{O}\left((1/\epsilon)^{2+\frac{\log(\kappa S)}{\log(1/\gamma)}}\right)$$

Complexity of uniform planning:

$$\mathcal{O}\left((1/\epsilon)^{2+\frac{\log(AS)}{\log(1/\gamma)}}\right)$$

Sample complexity bound of `TrailBlazer`

$$\mathcal{O}\left((1/\epsilon)^{\max\left(2,\frac{\log(\kappa S)}{\log(1/\gamma)}\right)}\right)$$

- From $+$ to max.
- Computationally efficient.

Planning with unbounded number of states

What if the MDP has a large state space (S is large or even infinite) ?

Planning with unbounded number of states

What if the MDP has a large state space (S is large or even infinite) ?

Uniform planning: [Kearns et al, 1999]

Sample complexity bound:

$$(1/\epsilon)^{\log(1/\epsilon)/\log(1/\gamma)}$$

Planning with unbounded number of states

What if the MDP has a large state space (S is large or even infinite) ?

Uniform planning: [Kearns et al, 1999]

Sample complexity bound:

$$(1/\epsilon)^{\log(1/\epsilon)/\log(1/\gamma)}$$

Adaptive planning: [Walsh et al, 2010]

Still no polynomial bound.

S-independent sample complexity bound

The sample complexity of the same algorithm `TrailBlazer` is bounded by:

$$\mathcal{O}\left((1/\epsilon)^{2+d}\right)$$

S-independent sample complexity bound

The sample complexity of the same algorithm `TrailBlazer` is bounded by:

$$\mathcal{O}\left((1/\epsilon)^{2+d}\right)$$

- The bound is independent of S.

S-independent sample complexity bound

The sample complexity of the same algorithm `TrailBlazer` is bounded by:

$$\mathcal{O}\left((1/\epsilon)^{2+d}\right)$$

- The bound is independent of S.
- Like κ, the quantity d is problem dependent.

S-independent sample complexity bound

The sample complexity of the same algorithm `TrailBlazer` is bounded by:

$$\mathcal{O}\left((1/\epsilon)^{2+d}\right)$$

- The bound is independent of S.
- Like κ, the quantity d is problem dependent.
- Unlike κ, the quantity d may be infinite.

S-independent sample complexity bound

The sample complexity of the same algorithm `TrailBlazer` is bounded by:

$$\mathcal{O}\left((1/\epsilon)^{2+d}\right)$$

- The bound is independent of S.
- Like κ, the quantity d is problem dependent.
- Unlike κ, the quantity d may be infinite.

Worst case: uniform planning using sparse sampling

$$(1/\epsilon)^{\log(1/\epsilon)/\log(1/\gamma)}$$

S-independent sample complexity bound

The sample complexity of the same algorithm `TrailBlazer` is bounded by:

$$\mathcal{O}\left((1/\epsilon)^{2+d}\right)$$

- The bound is independent of S.
- Like κ, the quantity d is problem dependent.
- Unlike κ, the quantity d may be infinite.

Worst case: uniform planning using sparse sampling

$$(1/\epsilon)^{\log(1/\epsilon)/\log(1/\gamma)}$$

When d is finite: polynomial S-independent bound.

Example of $d = 0$

Definition: Gap

The **gap** of a max node: difference between the best and the second best children values.
Low gap $\rightarrow$ difficult problems.

Example of $d = 0$

Definition: Gap

The **gap** of a max node: difference between the best and the second best children values.
Low gap $\to$ difficult problems.

$$\Delta(s) := \quad \text{average node } s \to \text{the gap of } s' \text{ with probability } p(s'|s).$$

Example of $d = 0$

Definition: Gap

The **gap** of a max node: difference between the best and the second best children values.
Low gap $\rightarrow$ difficult problems.

$$\Delta(s) := \quad \text{average node } s \rightarrow \text{the gap of } s' \text{ with probability } p(s'|s).$$

Assumption

$\exists a, b > 0$ s.t. for all average node s and $t > 0$

$$\mathbb{P}\left[\Delta(s) < t\right] < at^{2+b}$$

Small number of low gap nodes $\implies$ $\underline{d = 0}$.

Example of $d = 0$

Definition: Gap

The **gap** of a max node: difference between the best and the second best children values.
Low gap $\rightarrow$ difficult problems.

$$\Delta(s) := \quad \text{average node } s \rightarrow \text{the gap of } s' \text{ with probability } p(s'|s).$$

Assumption

$\exists a, b > 0$ s.t. for all average node s and $t > 0$

$$\mathbb{P}\left[\Delta(s) < t\right] < at^{2+b}$$

Small number of low gap nodes $\implies$ $\underline{d = 0}$.

Sample complexity of order $(1/\epsilon)^2$, same as **Monte Carlo sampling**.

Future work

We exhibit a class of problem for which the sample complexity of `TrailBlazer` is polynomial.
To our knowledge this is the first polynomial bound for planning in the S infinite case.

Future work

We exhibit a class of problem for which the sample complexity of `TrailBlazer` is polynomial.
To our knowledge this is the first polynomial bound for planning in the S infinite case.

Gap between the lower and upper bound for the worst case sample complexity of planning.

Future work

We exhibit a class of problem for which the sample complexity of `TrailBlazer` is polynomial.
To our knowledge this is the first polynomial bound for planning in the S infinite case.

Gap between the lower and upper bound for the worst case sample complexity of planning.

- Upper bound: non-polynomial

Future work

We exhibit a class of problem for which the sample complexity of `TrailBlazer` is polynomial.
To our knowledge this is the first polynomial bound for planning in the S infinite case.

Gap between the lower and upper bound for the worst case sample complexity of planning.

- Upper bound: non-polynomial
- Lower bound: polynomial

Conclusion

We introduce `TrailBlazer` a planning algorithm using sampling.

Conclusion

We introduce `TrailBlazer` a planning algorithm using sampling.

- `TrailBlazer` is easy to implement and computationally efficient.

Conclusion

We introduce `TrailBlazer` a planning algorithm using sampling.

• `TrailBlazer` is easy to implement and computationally efficient.

• In the worst case `TrailBlazer` performs the same as uniform planning with sparse sampling.

Conclusion

We introduce `TrailBlazer` a planning algorithm using sampling.

- `TrailBlazer` is easy to implement and computationally efficient.

- In the worst case `TrailBlazer` performs the same as uniform planning with sparse sampling.

- Finite S case: we improve over previous bounds.

Conclusion

We introduce `TrailBlazer` a planning algorithm using sampling.

• `TrailBlazer` is easy to implement and computationally efficient.

• In the worst case `TrailBlazer` performs the same as uniform planning with sparse sampling.

• Finite S case: we improve over previous bounds.

• Infinite S case: we highlight a class of problems for which `TrailBlazer` has polynomial complexity.

Conclusion

We introduce `TrailBlazer` a planning algorithm using sampling.

• `TrailBlazer` is easy to implement and computationally efficient.

• In the worst case `TrailBlazer` performs the same as uniform planning with sparse sampling.

• Finite S case: we improve over previous bounds.

• Infinite S case: we highlight a class of problems for which `TrailBlazer` has polynomial complexity.

`TrailBlazer` can be seen as a **natural extension of Monte Carlo Sampling to control problems.**

Thank You !

Poster number: 193