Improved Expected Running Time for MDP Planning

Shivaram Kalyanakrishnan[1] Neeldhara Misra[2] Aditya Gopalan[3]

1. Department of Computer Science and Engineering, Indian Institute of Technology Bombay
2. Department of Computer Science and Automation, Indian Institute of Science
3. Department of Electrical Communication Engineering, Indian Institute of Science

June 2015

Overview

1. MDP Planning
2. Solution strategies
   - Linear programming
   - Value iteration
   - Policy iteration
   - **Our contribution**: Planning by Guessing and Policy Improvement (PGPI)
3. PGPI algorithm
   - A total order on the set of policies
   - Guessing game
   - Algorithm
4. Discussion

Overview

1. MDP Planning
2. Solution strategies
   - Linear programming
   - Value iteration
   - Policy iteration
   - **Our contribution**: Planning by Guessing and Policy Improvement (PGPI)
3. PGPI algorithm
   - A total order on the set of policies
   - Guessing game
   - Algorithm
4. Discussion

MDP Planning

Markov Decision Problem: general abstraction of sequential decision making.

An MDP comprises a tuple (S, A, R, T), where

- S is a set of states (with $|S| = n$),
- A is a set of actions (with $|A| = k$),
- $R(s, a)$ is a bounded real number, $\forall s \in S, \forall a \in A$, and
- $T(s, a)$ is a probability distribution over S, $\forall s \in S, \forall a \in A$.

A policy $\pi : S \to A$ specifies an action from each state. The value of a policy π from state s is:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_t = \pi(s_t), t = 0, 1, 2, \ldots\right], \text{where}$$

$\gamma \in [0, 1)$ is a discount factor.

MDP Planning

Markov Decision Problem: general abstraction of sequential decision making.

An MDP comprises a tuple (S, A, R, T), where

- S is a set of states (with $|S| = n$),
- A is a set of actions (with $|A| = k$),
- $R(s, a)$ is a bounded real number, $\forall s \in S, \forall a \in A$, and
- $T(s, a)$ is a probability distribution over S, $\forall s \in S, \forall a \in A$.

A policy $\pi : S \to A$ specifies an action from each state. The value of a policy π from state s is:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_t = \pi(s_t), t = 0, 1, 2, \ldots\right], \text{where}$$

$\gamma \in [0, 1)$ is a discount factor.

Planning problem: Given S, A, R, T, and γ, find a policy $\pi^\star$ from the set of all policies Π such that

$$V^{\pi^\star}(s) \geq V^{\pi}(s), \forall s \in S, \forall \pi \in \Pi.$$

MDP Planning

Markov Decision Problem: general abstraction of sequential decision making.

An MDP comprises a tuple (S, A, R, T), where

- S is a set of states (with $|S| = n$),
- A is a set of actions (with $|A| = k$),
- $R(s, a)$ is a bounded real number, $\forall s \in S, \forall a \in A$, and
- $T(s, a)$ is a probability distribution over S, $\forall s \in S, \forall a \in A$.

A policy $\pi : S \to A$ specifies an action from each state. The value of a policy π from state s is:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_t \mid s_0 = s, a_t = \pi(s_t), t = 0, 1, 2, \ldots\right], \text{where}$$

$\gamma \in [0, 1)$ is a discount factor.

Planning problem: Given S, A, R, T, and γ, find a policy $\pi^{\star}$ from the set of all policies Π such that

$$V^{\pi^{\star}}(s) \geq V^{\pi}(s), \forall s \in S, \forall \pi \in \Pi.$$

Overview

1. MDP Planning
2. Solution strategies
   - Linear programming
   - Value iteration
   - Policy iteration
   - **Our contribution**: Planning by Guessing and Policy Improvement (PGPI)
3. PGPI algorithm
   - A total order on the set of policies
   - Guessing game
   - Algorithm
4. Discussion

Linear Programming

- The optimal value function $V^{\pi^\star} \stackrel{\text{def}}{=} V^\star$ is unique solution of Bellman's Optimality Equations: $\forall s \in S$:

$$V^\star(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\star(s') \right).$$

Linear Programming

- The optimal value function $V^{\pi^\star} \stackrel{\text{def}}{=} V^\star$ is unique solution of Bellman's Optimality Equations: $\forall s \in S$:

$$V^\star(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\star(s') \right).$$

- $V^\star$ can be obtained by solving an equivalent linear program:

$$\begin{aligned} &\text{maximise} && \sum_{s \in S} V(s) \\ &\text{subject to} && V(s) \geq \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right), \forall s \in S, \forall a \in A. \end{aligned}$$

Linear Programming

- The optimal value function $V^{\pi^\star} \stackrel{\text{def}}{=} V^\star$ is unique solution of Bellman's Optimality Equations: $\forall s \in S$:

$$V^\star(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\star(s') \right).$$

- $V^\star$ can be obtained by solving an equivalent linear program:

$$\text{maximise} \quad \sum_{s \in S} V(s)$$

$$\text{subject to} \quad V(s) \geq \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right), \forall s \in S, \forall a \in A.$$

- n variables, nk constraints (or *dual* with nk variables, n constraints).

Solution time: poly(n, k, B),

where B is the number of bits used to represent the MDP.

Value Iteration

- Classical dynamic programming approach.

$V_0 \leftarrow$ Arbitrary, element-wise bounded, n-length vector.
$t \leftarrow 0$.
Repeat:
  For $s \in S$**:**
    $V_{t+1}(s) \leftarrow \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_t(s') \right)$.
    $t \leftarrow t + 1$.
Until $V_t = V_{t-1} = V^\star$ (up to machine precision).

Value Iteration

- Classical dynamic programming approach.

$V_0 \leftarrow$ Arbitrary, element-wise bounded, n-length vector.
$t \leftarrow 0$.
Repeat:
  For $s \in S$**:**
    $V_{t+1}(s) \leftarrow \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_t(s') \right)$.
    $t \leftarrow t + 1$.
Until $V_t = V_{t-1} = V^\star$ (up to machine precision).

- Convergence to $V^\star$ guaranteed using a max-norm contraction argument.

Value Iteration

- Classical dynamic programming approach.

$V_0 \leftarrow$ Arbitrary, element-wise bounded, n-length vector.
$t \leftarrow 0$.
Repeat:
For $s \in S$**:**
$V_{t+1}(s) \leftarrow \max_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_t(s') \right)$.
$t \leftarrow t + 1$.
Until $V_t = V_{t-1} = V^\star$ (up to machine precision).

- Convergence to $V^\star$ guaranteed using a max-norm contraction argument.

Number of iterations: poly$(n, k, B, \frac{1}{1-\gamma})$.

Bellman's Equations and Policy Evaluation

Recall that

$$V^{\pi}(s) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | s_0 = s, a_i = \pi(s_i)].$$

Bellman's Equations ($\forall s \in S$):

$$\boxed{V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^{\pi}(s')\right].}$$

V^{π} is called the value function of π.

Bellman's Equations and Policy Evaluation

Recall that

$$V^{\pi}(s) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | s_0 = s, a_i = \pi(s_i)].$$

Bellman's Equations ($\forall s \in S$):

$$\boxed{V^{\pi}(s) = \textstyle\sum_{s' \in S} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^{\pi}(s')\right].}$$

V^{π} is called the value function of π.

Define ($\forall s \in S, \forall a \in A$):

$$Q^{\pi}(s, a) = \textstyle\sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi}(s')\right].$$

Q^{π} is called the action value function of π.

$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$.

Bellman's Equations and Policy Evaluation

Recall that

$$V^{\pi}(s) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | s_0 = s, a_i = \pi(s_i)].$$

Bellman's Equations ($\forall s \in S$):

$$\boxed{V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^{\pi}(s')\right].}$$

V^{π} is called the value function of π.

Define ($\forall s \in S, \forall a \in A$):

$$Q^{\pi}(s, a) = \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi}(s')\right].$$

Q^{π} is called the action value function of π.

$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$.

The variables in Bellman's equation are the $V^{\pi}(s)$. $|S|$ linear equations in $|S|$ unknowns.

Bellman's Equations and Policy Evaluation

Recall that

$$V^\pi(s) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | s_0 = s, a_i = \pi(s_i)].$$

Bellman's Equations ($\forall s \in S$):

$$V^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^\pi(s')\right].$$

V^π is called the value function of π.

Define ($\forall s \in S, \forall a \in A$):

$$Q^\pi(s, a) = \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^\pi(s')\right].$$

Q^π is called the action value function of π.

$V^\pi(s) = Q^\pi(s, \pi(s))$.

The variables in Bellman's equation are the $V^\pi(s)$. $|S|$ linear equations in $|S|$ unknowns.

Thus, given S, A, T, R, γ, and a fixed policy π, we can solve Bellman's equations efficiently to obtain, $\forall s \in S, \forall a \in A$, $V^\pi(s)$ and $Q^\pi(s, a)$.

Policy Iteration

- For a given policy π:

$$I(\pi) \stackrel{\text{def}}{=} \left\{ s \in S : Q^{\pi}(s, \pi(s)) < \max_{a \in A} Q^{\pi}(s, a) \right\}.$$

Policy Iteration

- For a given policy π:

$$I(\pi) \stackrel{\text{def}}{=} \left\{ s \in S : Q^{\pi}(s, \pi(s)) < \max_{a \in A} Q^{\pi}(s, a) \right\}.$$

- Q^{π} and I^{π} are easily derived from V^{π} (policy evaluation).

Policy Iteration

- For a given policy π:

$$I(\pi) \stackrel{\text{def}}{=} \left\{ s \in S : Q^{\pi}(s, \pi(s)) < \max_{a \in A} Q^{\pi}(s, a) \right\}.$$

- Q^{π} and I^{π} are easily derived from V^{π} (policy evaluation).

- $I^{\pi} = \emptyset$ iff π is an optimal policy.

Policy Iteration

- For a given policy π:

$$I(\pi) \stackrel{\text{def}}{=} \left\{ s \in S : Q^{\pi}(s, \pi(s)) < \max_{a \in A} Q^{\pi}(s, a) \right\}.$$

- Q^{π} and I^{π} are easily derived from V^{π} (policy evaluation).

- $I^{\pi} = \emptyset$ iff π is an optimal policy.

- Assume $I^{\pi} \neq \emptyset$. Let $C(\pi)$ be an arbitrarily "chosen" non-empty subset of $I(\pi)$.
- Define a policy π' as follows.

$$\pi'(s) \stackrel{\text{def}}{=} \begin{cases} \arg\max_{a \in A} Q^{\pi}(s, a) & \text{if } s \in C(\pi), \\ \pi(s) & \text{otherwise.} \end{cases}$$

Policy Iteration

- For a given policy π:

$$I(\pi) \stackrel{\text{def}}{=} \left\{ s \in S : Q^{\pi}(s, \pi(s)) < \max_{a \in A} Q^{\pi}(s, a) \right\}.$$

- Q^{π} and I^{π} are easily derived from V^{π} (policy evaluation).

- $I^{\pi} = \emptyset$ iff π is an optimal policy.

- Assume $I^{\pi} \neq \emptyset$. Let $C(\pi)$ be an arbitrarily "chosen" non-empty subset of $I(\pi)$.
- Define a policy π' as follows.

$$\pi'(s) \stackrel{\text{def}}{=} \begin{cases} \arg\max_{a \in A} Q^{\pi}(s, a) & \text{if } s \in C(\pi), \\ \pi(s) & \text{otherwise.} \end{cases}$$

- It can be shown that
  (1) $\forall s \in S : V^{\pi'}(s) \geq V^{\pi}(s)$, and
  (2) $\exists s \in S : V^{\pi'}(s) > V^{\pi}(s)$.

Policy Iteration

- For a given policy π:

$$I(\pi) \stackrel{\text{def}}{=} \left\{ s \in S : Q^{\pi}(s, \pi(s)) < \max_{a \in A} Q^{\pi}(s, a) \right\}.$$

- Q^{π} and I^{π} are easily derived from V^{π} (policy evaluation).

- $I^{\pi} = \emptyset$ iff π is an optimal policy.

- Assume $I^{\pi} \neq \emptyset$. Let $C(\pi)$ be an arbitrarily "chosen" non-empty subset of $I(\pi)$.
- Define a policy π' as follows.

$$\pi'(s) \stackrel{\text{def}}{=} \begin{cases} \arg\max_{a \in A} Q^{\pi}(s, a) & \text{if } s \in C(\pi), \\ \pi(s) & \text{otherwise.} \end{cases}$$

- It can be shown that
(1) $\forall s \in S : V^{\pi'}(s) \geq V^{\pi}(s)$, and
(2) $\exists s \in S : V^{\pi'}(s) > V^{\pi}(s)$. [Policy improvement]

Policy Iteration

$\pi_0 \leftarrow$ Arbitrary policy.
$t \leftarrow 0$.
Repeat:
  Evaluate π^t; derive $I(\pi^t)$.
  If $I(\pi^t) \neq \emptyset$, select $C(\pi^t) \subset I(\pi^t)$ and improve π^t to π^{t+1}.
  $t \leftarrow t + 1$.
Until $I(\pi^{t-1}) = \emptyset$.

Policy Iteration

$\pi_0 \leftarrow$ Arbitrary policy.
$t \leftarrow 0$.
Repeat:
  Evaluate π^t; derive $I(\pi^t)$.
  If $I(\pi^t) \neq \emptyset$, select $C(\pi^t) \subset I(\pi^t)$ and improve π^t to π^{t+1}.
  $t \leftarrow t + 1$.
Until $I(\pi^{t-1}) = \emptyset$.

- Howard's Policy Iteration: $C(\pi) = I(\pi)$.

Number of iterations: $O\left(\frac{k^n}{n}\right)$.

Policy Iteration

$\pi_0 \leftarrow$ Arbitrary policy.
$t \leftarrow 0$.
Repeat:
Evaluate π^t; derive $I(\pi^t)$.
If $I(\pi^t) \neq \emptyset$, select $C(\pi^t) \subset I(\pi^t)$ and improve π^t to π^{t+1}.
$t \leftarrow t + 1$.
Until $I(\pi^{t-1}) = \emptyset$.

- Howard's Policy Iteration: $C(\pi) = I(\pi)$.

Number of iterations: $O\left(\frac{k^n}{n}\right)$.

- Mansour and Singh's Randomised Policy Iteration: $C(\pi)$ chosen uniformly at random from among the non-empty subsets of $I(\pi)$.

Expected number of iterations:
$O\left(2^{0.78n}\right)$ for k = 2; $O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ for general k.

Policy Iteration

$\pi_0 \leftarrow$ Arbitrary policy.
$t \leftarrow 0$.
Repeat:
  Evaluate π^t; derive $I(\pi^t)$.
  If $I(\pi^t) \neq \emptyset$, select $C(\pi^t) \subset I(\pi^t)$ and improve π^t to π^{t+1}.
  $t \leftarrow t + 1$.
Until $I(\pi^{t-1}) = \emptyset$.

- Howard's Policy Iteration: $C(\pi) = I(\pi)$.

Number of iterations: $O\left(\frac{k^n}{n}\right)$.

- Mansour and Singh's Randomised Policy Iteration: $C(\pi)$ chosen uniformly at random from among the non-empty subsets of $I(\pi)$.

Expected number of iterations:
$O\left(2^{0.78n}\right)$ for k = 2; $O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ for general k.

- References: Howard(1960), Mansour and Singh (1999), Hollanders et al. (2014).

Policy Iteration

$\pi_0 \leftarrow$ Arbitrary policy.
$t \leftarrow 0$.
Repeat:
  Evaluate π^t; derive $I(\pi^t)$.
  If $I(\pi^t) \neq \emptyset$, select $C(\pi^t) \subset I(\pi^t)$ and improve π^t to π^{t+1}.
  $t \leftarrow t + 1$.
Until $I(\pi^{t-1}) = \emptyset$.

- Howard's Policy Iteration: $C(\pi) = I(\pi)$.

Number of iterations: $O\left(\frac{k^n}{n}\right)$.

- Mansour and Singh's Randomised Policy Iteration: $C(\pi)$ chosen uniformly at random from among the non-empty subsets of $I(\pi)$.

Expected number of iterations:
$O\left(2^{0.78n}\right)$ for k = 2; $O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ for general k.

- References: Howard(1960), Mansour and Singh (1999), Hollanders et al. (2014).
- Note that bounds do not depend on B and γ!

Our Contribution

Algorithm	**Computational complexity**
Linear Programming	$\text{poly}(n, k, B)$
Value Iteration	$\text{poly}\left(n, k, B, \frac{1}{1-\gamma}\right)$
Policy Iteration	$\text{poly}(n) \cdot O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ (expected)

Our Contribution

Algorithm	**Computational complexity**
Linear Programming	$\text{poly}(n, k, B)$
Value Iteration	$\text{poly}\left(n, k, B, \frac{1}{1-\gamma}\right)$
Policy Iteration	$\text{poly}(n) \cdot O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ (expected)
PGPI	$\text{poly}(n) \cdot O\left(k^{\frac{n}{2}}\right)$ (expected)

Our Contribution

Algorithm	Computational complexity
Linear Programming	$\text{poly}(n, k, B)$
Value Iteration	$\text{poly}\left(n, k, B, \frac{1}{1-\gamma}\right)$
Policy Iteration	$\text{poly}(n) \cdot O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ (expected)
PGPI	$\text{poly}(n) \cdot O\left(k^{\frac{n}{2}}\right)$ (expected)

PGPI = Planning by Guessing and Policy Improvement.

Randomised algorithm.

Key ingredient: a **total order** on the set of policies.

Analysis involves basic probability and counting.

Our Contribution

Algorithm	**Computational complexity**
Linear Programming	$\text{poly}(n, k, B)$
Value Iteration	$\text{poly}\left(n, k, B, \frac{1}{1-\gamma}\right)$
Policy Iteration	$\text{poly}(n) \cdot O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ (expected)
PGPI	$\text{poly}(n) \cdot O\left(k^{\frac{n}{2}}\right)$ (expected)

PGPI = Planning by Guessing and Policy Improvement.

Randomised algorithm.

Key ingredient: a **total order** on the set of policies.

Analysis involves basic probability and counting.

Even tighter bounds by combining PGPI with Randomised Policy Iteration!

Our Contribution

Algorithm	Computational complexity
Linear Programming	poly(n, k, B) [“subexponential” bounds (n, k) exist!]
Value Iteration	$\text{poly}\left(n, k, B, \frac{1}{1-\gamma}\right)$
Policy Iteration	$\text{poly}(n) \cdot O\left(\left(\left(1 + \frac{2}{log(k)}\right)\frac{k}{2}\right)^n\right)$ (expected)
PGPI	$\text{poly}(n) \cdot O\left(k^{\frac{n}{2}}\right)$ (expected)

PGPI = Planning by Guessing and Policy Improvement.

Randomised algorithm.

Key ingredient: a **total order** on the set of policies.

Analysis involves basic probability and counting.

Even tighter bounds by combining PGPI with Randomised Policy Iteration!

Overview

1. MDP Planning

2. Solution strategies
   - Linear programming
   - Value iteration
   - Policy iteration
   - **Our contribution**: Planning by Guessing and Policy Improvement (PGPI)

3. PGPI algorithm
   - A total order on the set of policies
   - Guessing game
   - Algorithm

4. Discussion

A Total Order on the Set of Policies

- For $\pi \in \Pi$, define

$$V(\pi) = \sum_{s \in S} V^{\pi}(s).$$

A Total Order on the Set of Policies

- For $\pi \in \Pi$, define
$$V(\pi) = \sum_{s \in S} V^{\pi}(s).$$

- Let L be an arbitrary total ordering on policies for tie-breaking, for example the lexicographic ordering.

A Total Order on the Set of Policies

- For $\pi \in \Pi$, define
$$V(\pi) = \sum_{s \in S} V^{\pi}(s).$$

- Let L be an arbitrary total ordering on policies for tie-breaking, for example the lexicographic ordering.

- Total order $\succ$:

> For $\pi_1, \pi_2 \in \Pi$, define $\pi_1 \succ \pi_2$ iff
> $V(\pi_1) > V(\pi_2)$ or
> $V(\pi_1) = V(\pi_2)$ and $\pi_1 L \pi_2$.

A Total Order on the Set of Policies

- For $\pi \in \Pi$, define

$$V(\pi) = \sum_{s \in S} V^{\pi}(s).$$

- Let L be an arbitrary total ordering on policies for tie-breaking, for example the lexicographic ordering.

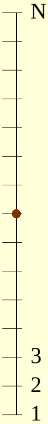
- Total order $\succ$:

For $\pi_1, \pi_2 \in \Pi$, define $\pi_1 \succ \pi_2$ iff
$V(\pi_1) > V(\pi_2)$ or
$V(\pi_1) = V(\pi_2)$ and $\pi_1 L \pi_2$.

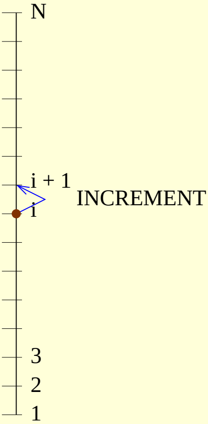
- Observe that if policy improvement to π yields π', then $\pi' \succ \pi$.

$$\forall s \in S : V^{\pi'}(s) \geq V^{\pi}(s) \text{ and } \exists s \in S : V^{\pi'}(s) > V^{\pi}(s)$$
$$\implies V(\pi') > V(\pi)$$
$$\implies \pi_1 \succ \pi_2.$$

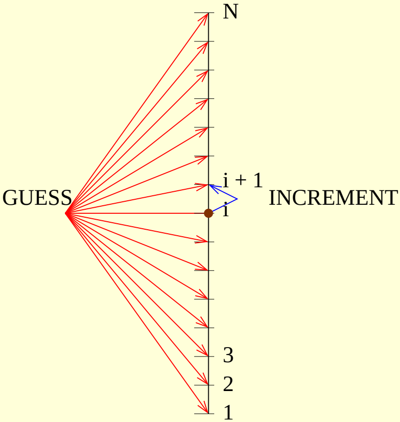
Guessing Game



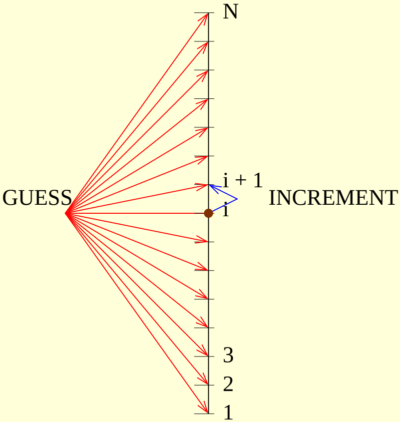
Guessing Game



Guessing Game



Guessing Game

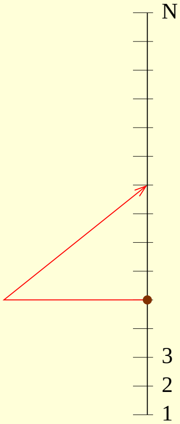


- How many operations needed to reach N?

Guessing Game

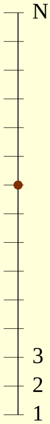
- How many operations needed to reach N?

Guessing Game



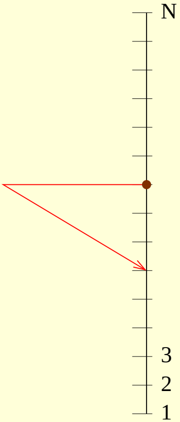
- How many operations needed to reach N?

Guessing Game



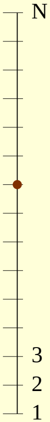
- How many operations needed to reach N?

Guessing Game



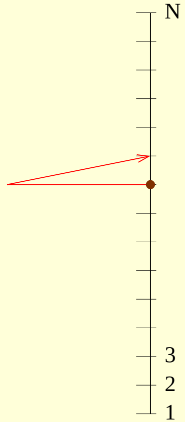
- How many operations needed to reach N?

Guessing Game



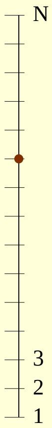
- How many operations needed to reach N?

Guessing Game



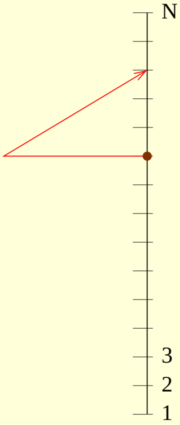
- How many operations needed to reach N?

Guessing Game



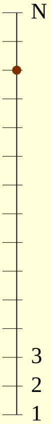
- How many operations needed to reach N?

Guessing Game



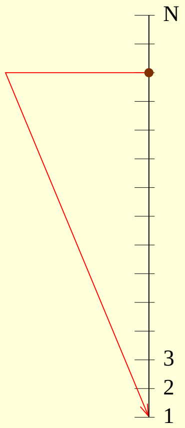
- How many operations needed to reach N?

Guessing Game



- How many operations needed to reach N?

Guessing Game

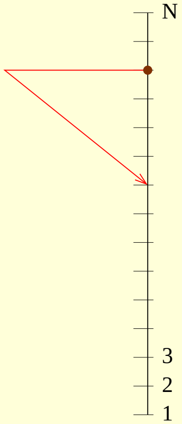


- How many operations needed to reach N?

Guessing Game

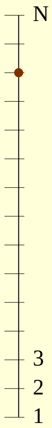
- How many operations needed to reach N?

Guessing Game



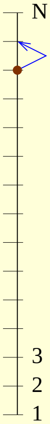
- How many operations needed to reach N?

Guessing Game



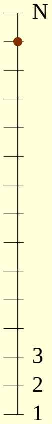
- How many operations needed to reach *N*?

Guessing Game



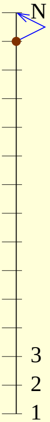
- How many operations needed to reach N?

Guessing Game



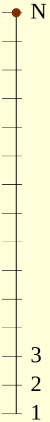
- How many operations needed to reach N?

Guessing Game



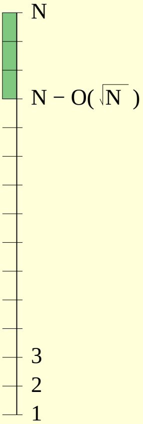
- How many operations needed to reach N?

Guessing Game



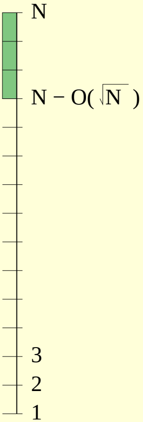
- How many operations needed to reach N?

Guessing Game



- How many operations needed to reach N?
- Pick the best of $\sqrt{N}$ guesses, and then increment up to N.

Guessing Game



- How many operations needed to reach N?
- Pick the best of $\sqrt{N}$ guesses, and then increment up to N.
- Expected number of operations: $O(\sqrt{N})$.

PGPI Algorithm

$\pi \leftarrow$ Arbitrary policy.
Repeat $k^{\alpha n}$ times:
  Draw π' from Π uniformly at random.
  If $\pi' \succ \pi$, $\pi \leftarrow \pi'$.
While π is not optimal:
  $\pi \leftarrow$ PolicyImprovement(π).

PGPI Algorithm

$\pi \leftarrow$ Arbitrary policy.
Repeat $k^{\alpha n}$ times:
  Draw π' from Π uniformly at random.
  If $\pi' \succ \pi$, $\pi \leftarrow \pi'$.
While π is not optimal:
  $\pi \leftarrow$ PolicyImprovement(π).

$\alpha = 0.5$; **Expected number of iterations**: $O\left(k^{\frac{n}{2}}\right)$.

PGPI Algorithm

$\pi \leftarrow$ Arbitrary policy.
Repeat $k^{\alpha n}$ times:
    Draw π' from Π uniformly at random.
    If $\pi' \succ \pi$, $\pi \leftarrow \pi'$.
While π is not optimal:
    $\pi \leftarrow$ PolicyImprovement(π).

$\alpha = 0.5$; **Expected number of iterations**: $O\left(k^{\frac{n}{2}}\right)$.

$k = 2, \alpha = 0.46$, randomised policy improvement;
Expected number of iterations: $O\left(2^{0.46n}\right)$.

Overview

1. MDP Planning
2. Solution strategies
   - Linear programming
   - Value iteration
   - Policy iteration
   - **Our contribution**: Planning by Guessing and Policy Improvement (PGPI)
3. PGPI algorithm
   - A total order on the set of policies
   - Guessing game
   - Algorithm
4. Discussion

Discussion

- PGPI: improved complexity bound solely in terms of n and k.

Discussion

- PGPI: improved complexity bound solely in terms of n and k.
- Policy Iteration:MDP Planning :: Simplex: Linear Programming? Connections?

Discussion

- PGPI: improved complexity bound solely in terms of n and k.

- Policy Iteration:MDP Planning :: Simplex: Linear Programming? Connections?

- PGPI: no favourable experimental results yet!
- Policy Iteration (Howard, Mansour and Singh) very quick on “typical” MDPs.
- Yet to find MDPs on which PGPI dominates Policy Iteration.
- Currently known “lower bound” MDPs (Fearnley, 2010) not suitable.

Discussion

- PGPI: improved complexity bound solely in terms of n and k.

- Policy Iteration:MDP Planning :: Simplex: Linear Programming? Connections?

- PGPI: no favourable experimental results yet!
- Policy Iteration (Howard, Mansour and Singh) very quick on "typical" MDPs.
- Yet to find MDPs on which PGPI dominates Policy Iteration.
- Currently known "lower bound" MDPs (Fearnley, 2010) not suitable.

- References.

R. A. Howard, 1960. Dynamic Programming and Markov Processes. MIT Press, 1960.

Yishay Mansour and Satinder Singh, 1999. On the Complexity of Policy Iteration. *In Proc. UAI 1999*, pp. 401–408, AUAI, 1999.

John Fearnley, 2010. Exponential Lower Bounds for Policy Iteration. *In Proc. ICALP 2010*, pp. 551–562, Springer, 2010.

Romain Hollanders, Balázs Gerencsér, Jean-Charles Delvenne, and Raphaël M. Jungers, 2014. About upper bounds on the complexity of Policy Iteration, `http://perso.uclouvain.be/romain.hollanders/docs/NewUpperBoundForPI.pdf`.

Discussion

- PGPI: improved complexity bound solely in terms of n and k.

- Policy Iteration:MDP Planning :: Simplex: Linear Programming? Connections?

- PGPI: no favourable experimental results yet!
- Policy Iteration (Howard, Mansour and Singh) very quick on “typical” MDPs.
- Yet to find MDPs on which PGPI dominates Policy Iteration.
- Currently known “lower bound” MDPs (Fearnley, 2010) not suitable.

- References.

R. A. Howard, 1960. Dynamic Programming and Markov Processes. MIT Press, 1960.

Yishay Mansour and Satinder Singh, 1999. On the Complexity of Policy Iteration. *In Proc. UAI 1999*, pp. 401–408, AUAI, 1999.

John Fearnley, 2010. Exponential Lower Bounds for Policy Iteration. *In Proc. ICALP 2010*, pp. 551–562, Springer, 2010.

Romain Hollanders, Balázs Gerencsér, Jean-Charles Delvenne, and Raphaël M. Jungers, 2014. About upper bounds on the complexity of Policy Iteration, `http://perso.uclouvain.be/romain.hollanders/docs/NewUpperBoundForPI.pdf`.

Thank you!