Hierarchical Width-Based Planning and Learning

Miquel Junyent, Vicenç Gómez, Anders Jonsson

Universitat Pompeu Fabra, Barcelona, Spain

Universitat Pompeu Fabra Barcelona

ICAPS
August 2021

IW(w) (Lipovetzky and Geffner 2012)

- Original algorithm: **Breadth-first search** (BrFS) method
- Requires states to be factored into **features** $\phi(s)$
- Prunes states that are not **novel for width *w***
- A state is **novel** if it has a **feature tuple of size *w*** that is **new in the search**
- **Complexity exponential in *w***, but independent of $|\mathcal{S}|$
- Most classical planning benchmarks present a **low width** when the **goal is a single atom**.

# Domains	# Inst.	Inst. IW(1)	Inst. IW(2)
37	37,921	37.0%	88.2%

Summary of Table 1 (Lipovetzky and Geffner 2012)

IW(*w*) (Lipovetzky and Geffner 2012)

- Original algorithm: **Breadth-first search** (BrFS) method
- Requires states to be factored into **features** $\phi(s)$
- Prunes states that are not **novel for width *w***
- A state is **novel** if it has a **feature tuple of size *w*** that is **new in the search**
- **Complexity exponential in *w***, but independent of $|\mathcal{S}|$
- Most classical planning benchmarks present a **low width** when the **goal is a single atom**.
- In practice:
  - Problems have **higher width** (no single-atom goal tasks)
  - IW(w) is mostly used with **w=1** due to computational constraints

Complexity of IW(w)

- Let $N(n, d, w)$ denote the **maximum amount of novel nodes** that $\mathrm{IW}(w)$ generates in a problem with $n = |F|$ features of domain size $d = |D|$
- Our result is based on two basic premises:
  - A feature **has one value** at a time
  - A feature value **appears in several tuples** simultaneously
- Recursive formula

Complexity of IW(w)

- Let $N(n, d, w)$ denote the **maximum amount of novel nodes** that $\mathrm{IW}(w)$ generates in a problem with $n = |F|$ features of domain size $d = |D|$
- Our result is based on two basic premises:
  - A feature **has one value** at a time
  - A feature **appears in several tuples** simultaneously
- Recursive formula

$$N(n, d, 0) = 1,$$

Only the initial state is novel

$w = 0$

Complexity of IW(w)

- Let $N(n, d, w)$ denote the **maximum amount of novel nodes** that $\mathrm{IW}(w)$ generates in a problem with $n = |F|$ features of domain size $d = |D|$
- Our result is based on two basic premises:
  - A feature **has one value** at a time
  - A feature **appears in several tuples** simultaneously
- Recursive formula

$$N(n, d, 0) = 1,$$

$$N(n, d, n) = d^n, \longrightarrow \boxed{\text{All states are novel}}$$

$$\boxed{n = w}$$

Complexity of IW(w)

- Let $N(n, d, w)$ denote the **maximum amount of novel nodes** that $\mathrm{IW}(w)$ generates in a problem with $n = |F|$ features of domain size $d = |D|$
- Our result is based on two basic premises:
  - A feature **has one value** at a time
  - A feature **appears in several tuples** simultaneously
- Recursive formula

$$
\begin{aligned}
N(n, d, 0) &= 1, \\
N(n, d, n) &= d^n, \\
N(n, d, w) &= \underbrace{(d-1)N(n-1, d, w-1)}_{\text{States novel due to one feature } f} + \underbrace{N(n-1, d, w)}_{\text{States novel due to other features different than } f}.
\end{aligned}
$$

Complexity of IW(w)

- Let $N(n, d, w)$ denote the **maximum amount of novel nodes** that $\mathrm{IW}(w)$ generates in a problem with $n = |F|$ features of domain size $d = |D|$
- Our result is based on two basic premises:
  - A feature **has one value** at a time
  - A feature **appears in several tuples** simultaneously
- General formula, for $0 \leq w < n$:

$$N(n, d, w) = \sum_{k=0}^{w} \left[\binom{n-1-k}{w-k} d^k (d-1)^{w-k} \right].$$

A hierarchical approach to blind search

- Blind search methods require two components:
  - **Successor function**: given a state and an action, returns a successor state (e.g., simulator)
  - **Stopping condition**: tells us when to stop the search (e.g., goal is met)

Example BrFS:

```
Q = Queue(root)
While Q not empty:
    s = PopFirst(Q)
    For each action a:
        x = GenerateSuccessor(s,a)
        Append(Q, x)
        If ShouldStop(x):
            return
```

A hierarchical approach to blind search

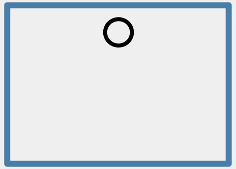
- Our **hierarchical approach**:
  - Considers two sets of features F_h and F_ℓ

A state is represented by its feature vector $\phi_\ell(s)$

A hierarchical approach to blind search

- Our **hierarchical approach**:
  - Considers two sets of features F_h and F_ℓ

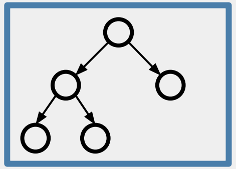
It also belongs to a **high-level state**, represented by the feature mapping $\phi_h(s)$



A hierarchical approach to blind search

- Our **hierarchical approach**:
  - Considers two sets of features F_h and F_ℓ
  - Modifies:
    - **High-level successor function**: each call to this function triggers a low-level search

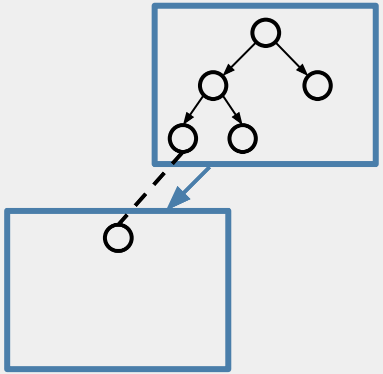
Each **high-level state** contains a **low-level tree,** where states share the same $\phi_h(s)$



A hierarchical approach to blind search

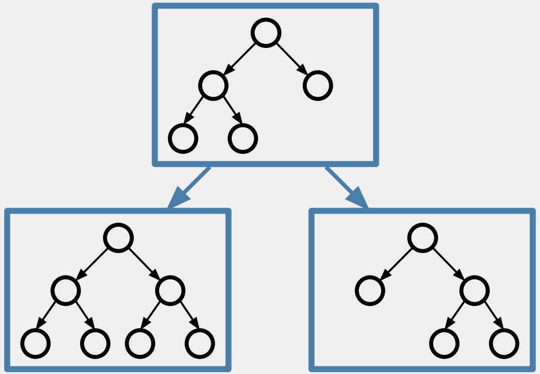
- Our **hierarchical approach**:
  - Considers two sets of features F_h and F_ℓ
  - Modifies:
    - **High-level successor function**: each call to this function triggers a low-level search
    - **Low-level stopping condition**: stops the low-level search when a state s that maps to a different $\phi_h(s)$ is found

When a state maps to a **different** $\phi_h(s)$, the low-level search is stopped and a **new high-level node** is created



A hierarchical approach to blind search

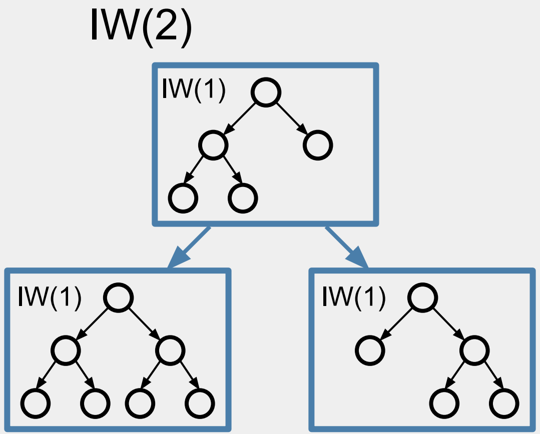
- Our **hierarchical approach**:
  - Considers two sets of features F_h and F_ℓ
  - Modifies:
    - **High-level successor function**: each call to this function triggers a low-level search
    - **Low-level stopping condition**: stops the low-level search when a state s that maps to a different $\phi_h(s)$ is found
  - We can pause and **resume low-level searches**
  - Allows for **many levels of abstraction**
  - Accepts **different planners** at each level



Hierarchical IW

- We can use **IW(*w*) at the different levels**
- For instance:
  - IW(2) at high-level
  - IW(1) at low-level

$\left.\begin{array}{l}\text{IW(2) at high-level}\\ \text{IW(1) at low-level}\end{array}\right\}$ HIW(2, 1)

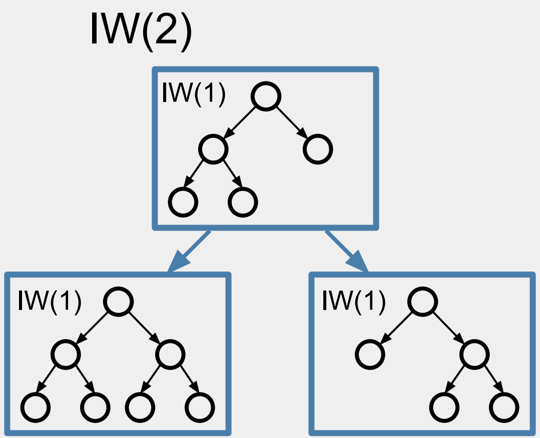


Hierarchical IW

- We can use **IW(*w*) at the different levels**
- For instance:
  - IW(2) at high-level
  - IW(1) at low-level

  HIW(2, 1)
- In general: $\mathrm{HIW}(w_h, w_\ell)$
- HIW can solve problems of width $w_h + w_\ell$

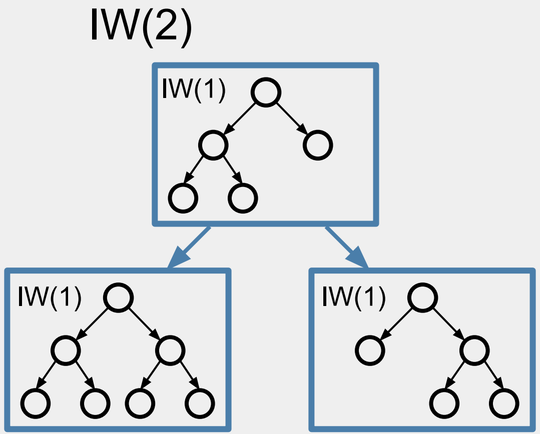
Features:
- 1-D position
- Having the key



Hierarchical IW

- We can use **IW(w) at the different levels**
- For instance:
    - IW(2) at high-level
    - IW(1) at low-level

  HIW(2, 1)
- In general: $\mathrm{HIW}(w_h, w_\ell)$
- HIW can solve problems of width $w_h + w_\ell$



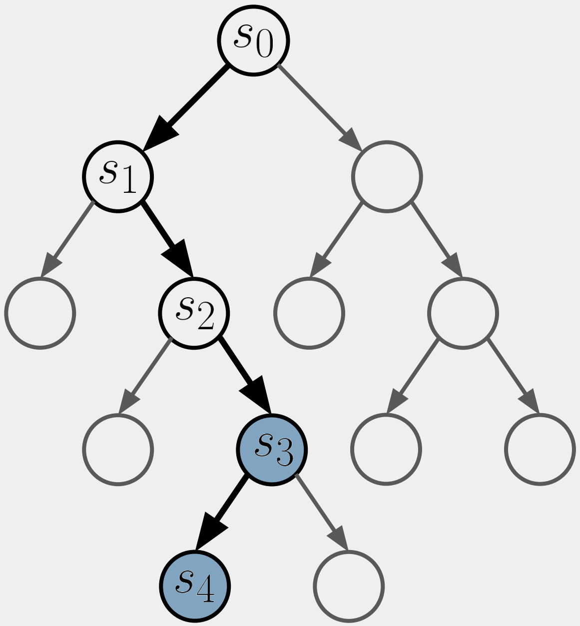
Features:

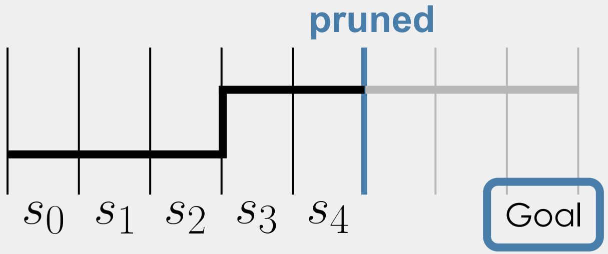
- 1-D position **(low level)**
- Having the key **(high level)**

This problem has **width 2**, but it can be solved by **HIW(1,1)**

Incremental HIW

- **Hypothesis**: features that **only change once** in a branch before being pruned are good candidates

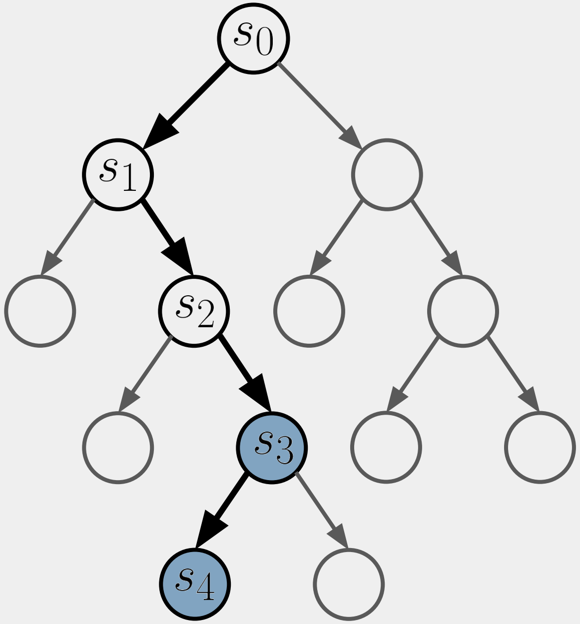


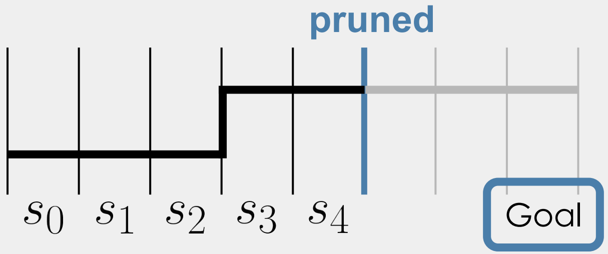


Incremental HIW

- **Hypothesis**: features that **only change once** in a branch before being pruned are good candidates

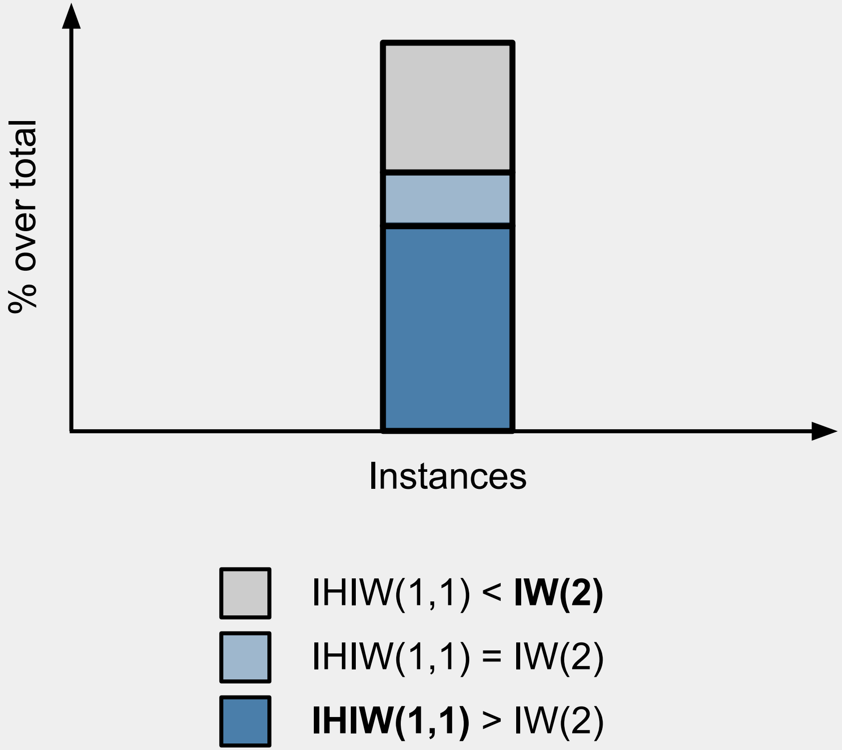
- **Incremental HIW(1,1):**
  - Iteratively **run HIW(1,1)**
  - **Add one feature** to F_h at each iteration
  - **Discover new features** when necessary
  - **Reuse** the search **tree** among iterations





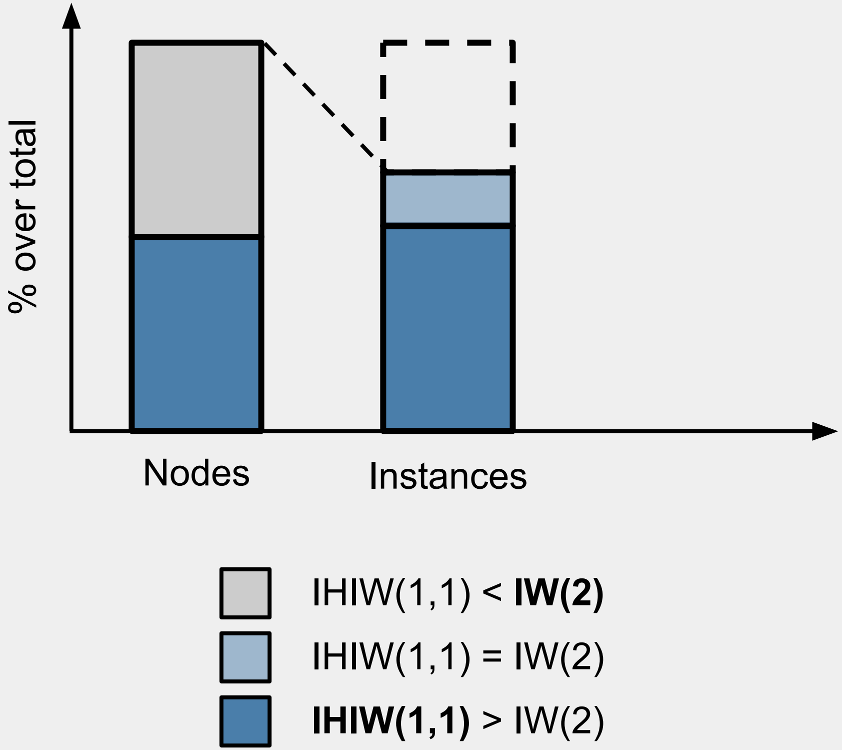
Results in classical planning

- Single goal instances
- Budget of 10K nodes
- We report:
  - Solved instances (%)
  - Avg. nodes (solved)
  - Avg. time (solved)
- **IHIW > IW(1) in 31/36** domains
- **IHIW ≥ IW(2) in 24/36** domains



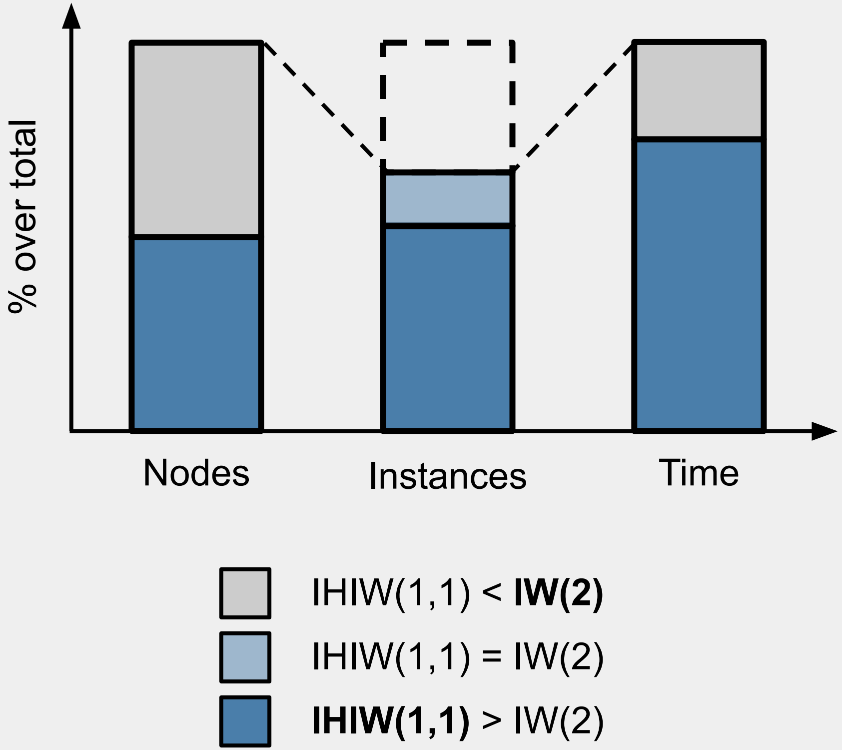
Results in classical planning

- Single goal instances
- Budget of 10K nodes
- We report:
  - Solved instances (%)
  - Avg. nodes (solved)
  - Avg. time (solved)
- **IHIW > IW(1) in all but 5 domains**
- **IHIW ≥ IW(2) in 24/36** domains
  - Uses **less nodes in 12/24**



Results in classical planning

- Single goal instances
- Budget of 10K nodes
- We report:
  - Solved instances (%)
  - Avg. nodes (solved)
  - Avg. time (solved)
- **IHIW > IW(1) in all but 5 domains**
- **IHIW ≥ IW(2) in 24/36** domains
  - Uses **less nodes in 12/24**
  - Solves it **faster in 18/24**



π-HIW

- Integrating HIW with a **policy learning** scheme

π-HIW

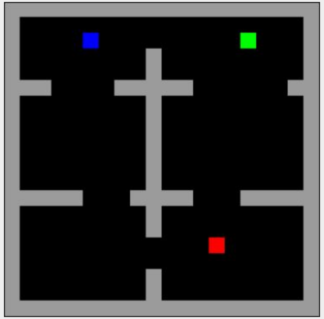
- Integrating HIW with a **policy learning** scheme
  - **High-level** planner: **Count-Based Rollout IW**
    - Selects high-level nodes according to $p \propto \exp\left(1/\tau(c+1)\right)$
    - Prunes nodes using a mapping from novel tuples to unpruned nodes

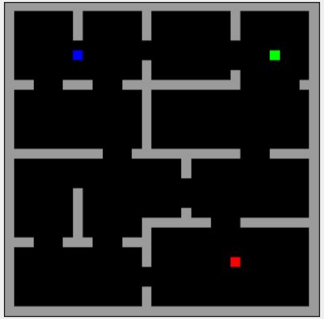
π-HIW

- Integrating HIW with a **policy learning** scheme
  - **High-level** planner: **Count-Based Rollout IW**
    - Selects high-level nodes according to $p \propto \exp\left(1/\tau(c+1)\right)$
    - Prunes nodes using a mapping from novel tuples to unpruned nodes
  - **Low-level** planner: **π-IW modified**
    - Tree counts for tie-breaking (when the reward is sparse)
    - Value function

Results in gridworld

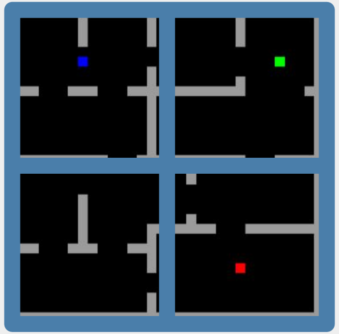
- Sparse reward tasks. The episode terminates:
  - when the agent picks the key and reaches the door (r = +1)
  - when hitting a wall (r = -1)
  - after 200 / 500 steps (r = 0)





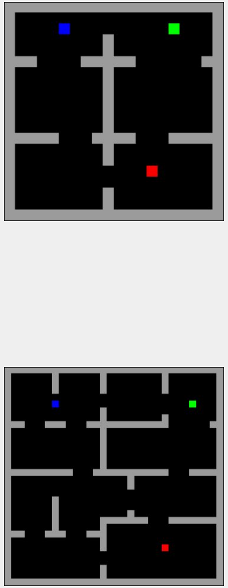
Results in gridworld

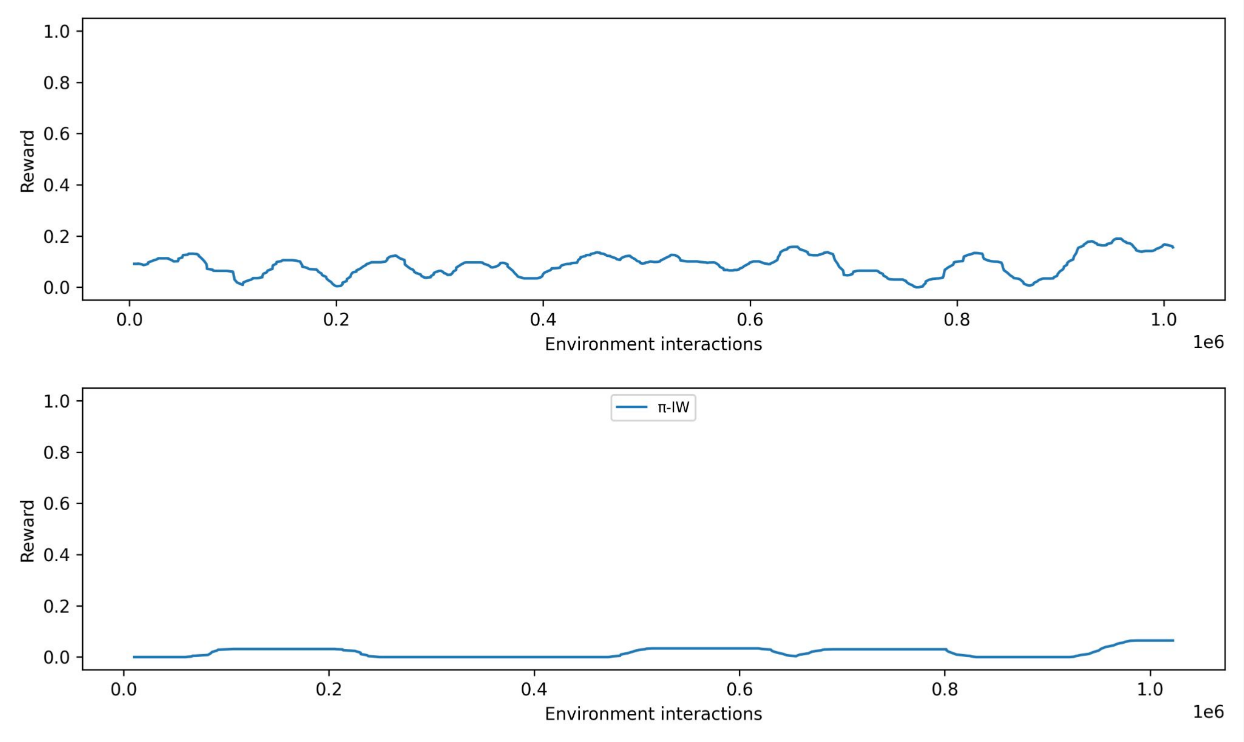
- Sparse reward tasks. The episode terminates:
  - when the agent picks the key and reaches the door (r = +1)
  - when hitting a wall (r = -1)
  - after 200 / 500 steps (r = 0)
- Features:
  - Neural network activations (low level)
  - Downsampling (high level)



Example: 2x2 tiles

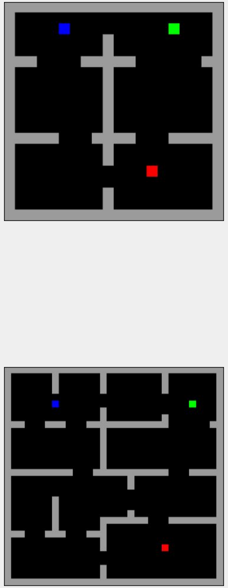
Results in gridworld

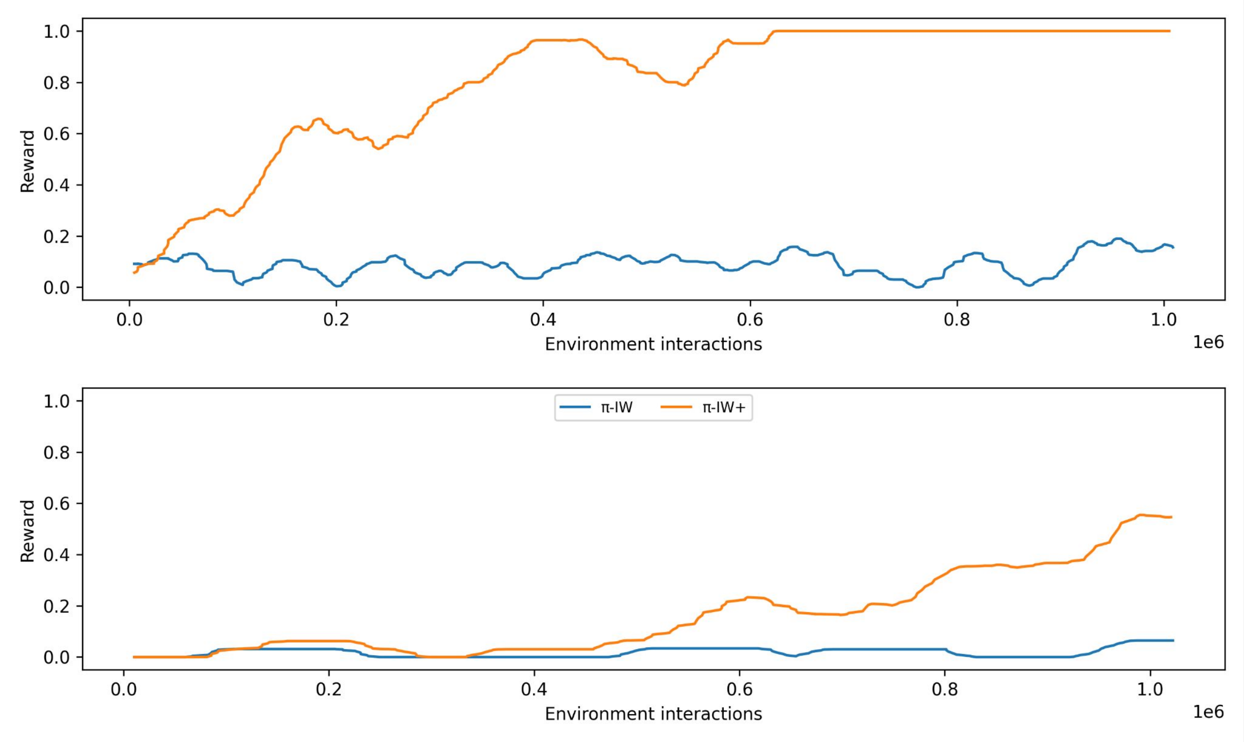




Features: neural network activations

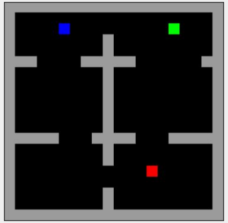
Results in gridworld

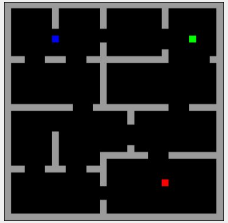


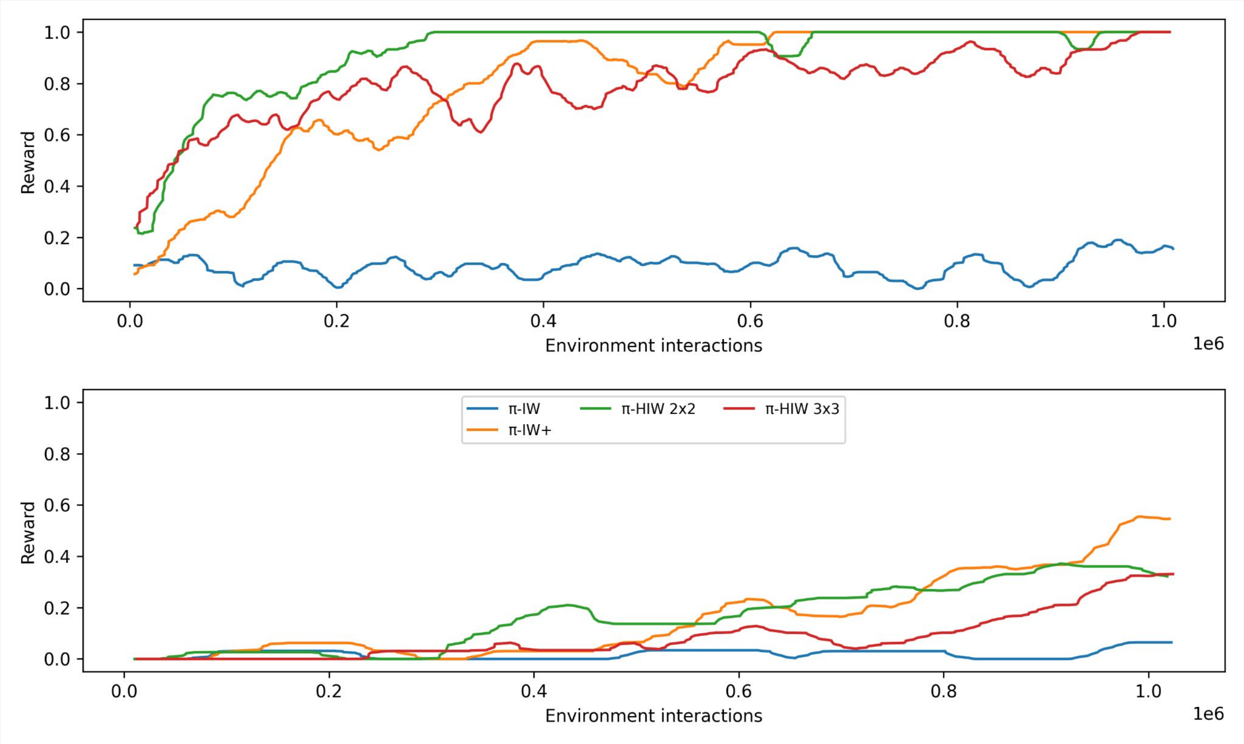


Features: neural network activations

Results in gridworld

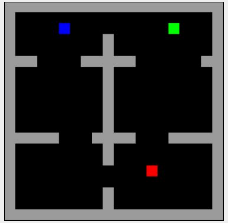


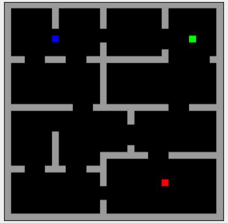


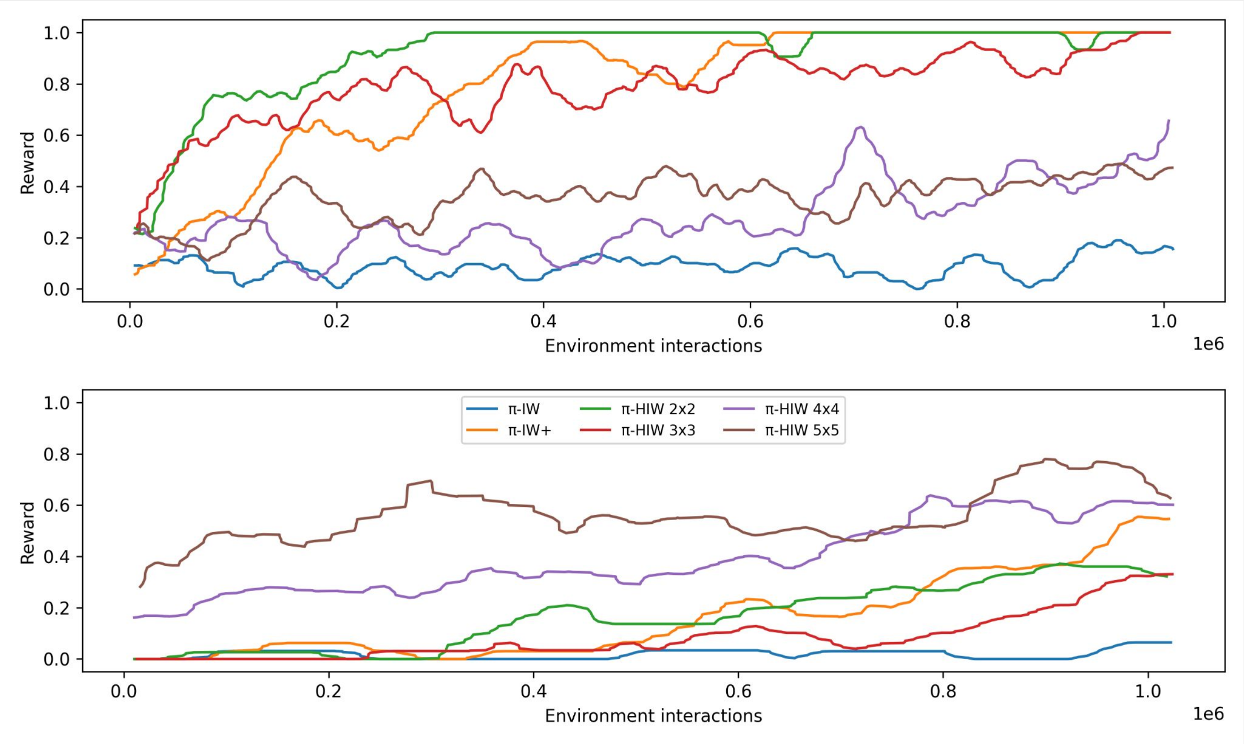


Features: neural network activations **(low level)**
downsampling **(high level)**

Results in gridworld

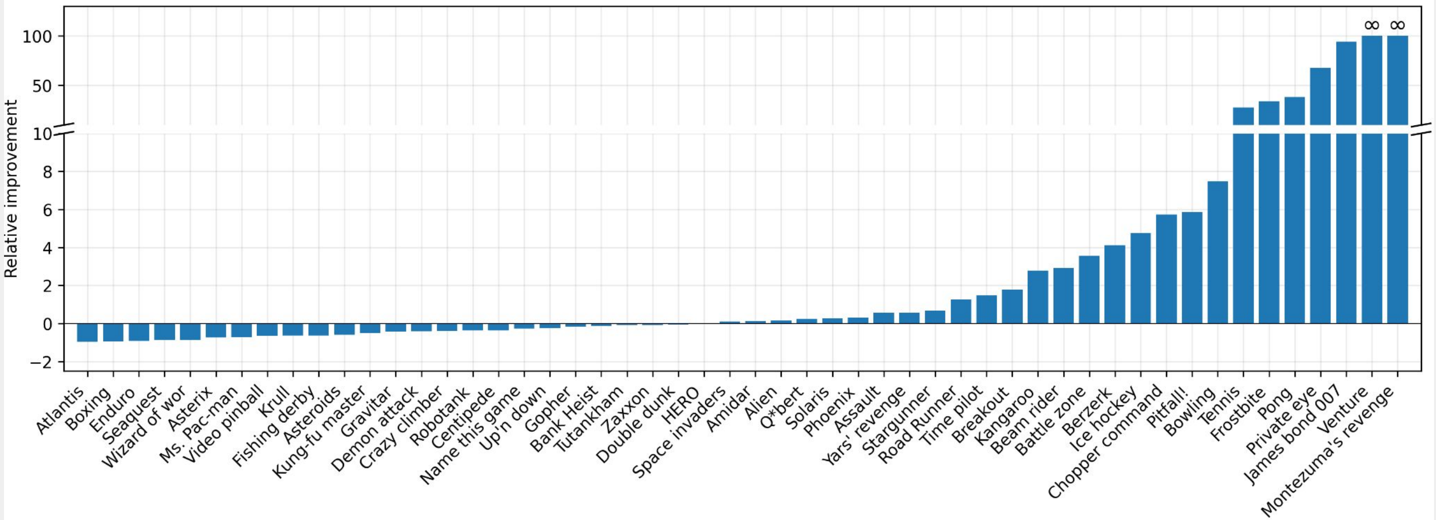




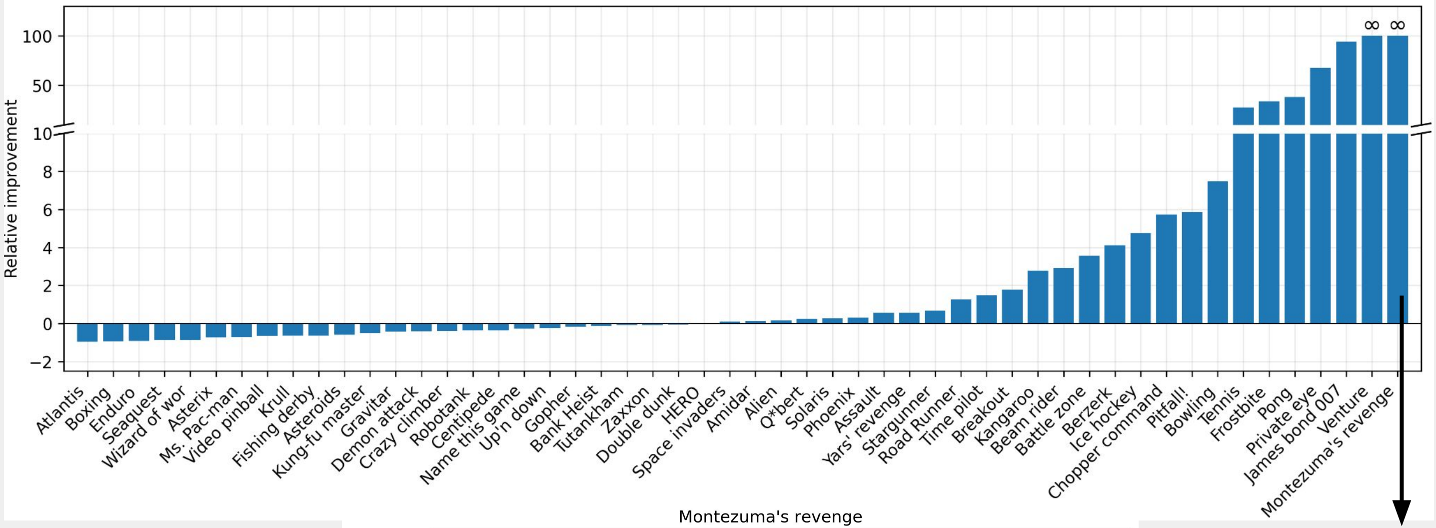


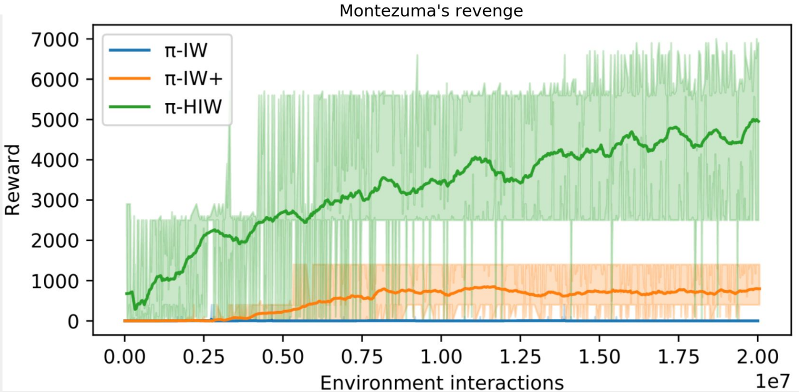
Features: neural network activations **(low level)**
downsampling **(high level)**

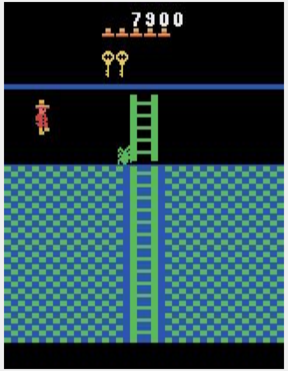
Results in Atari games



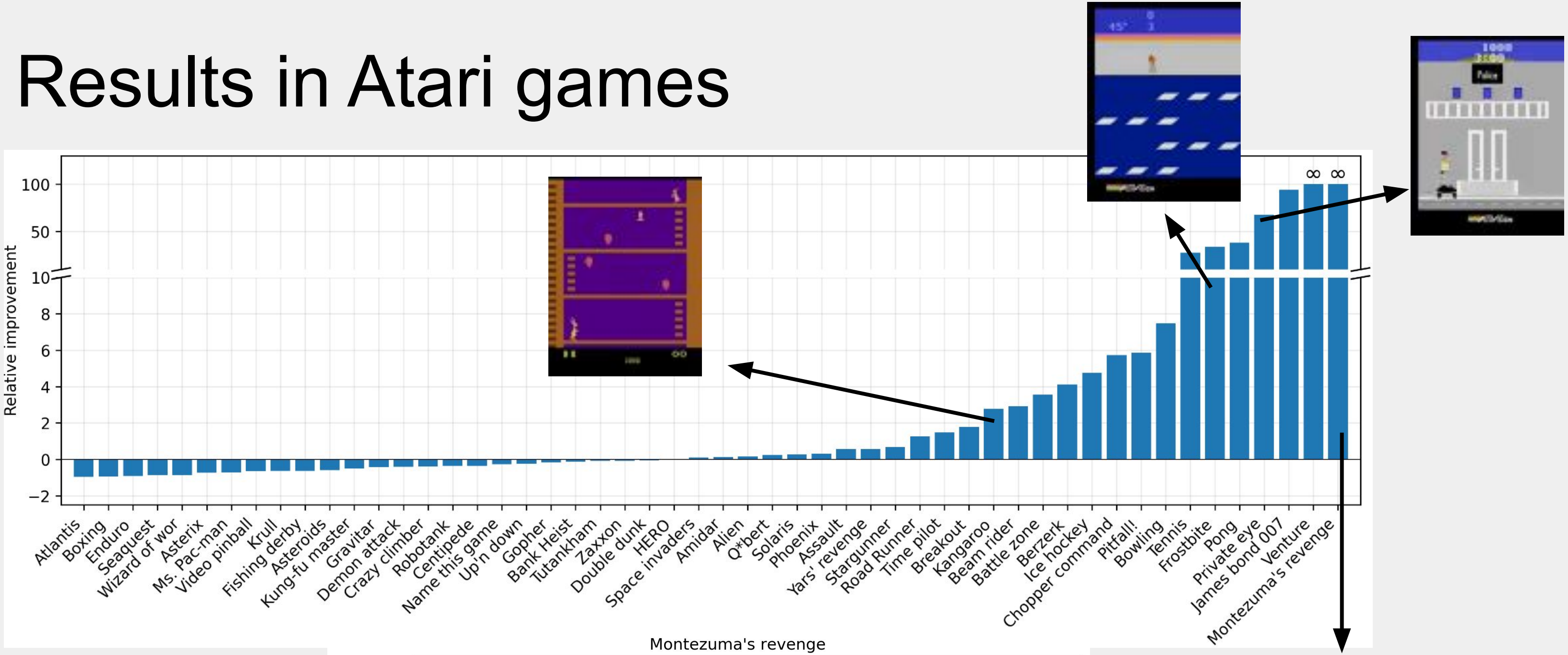
Results in Atari games

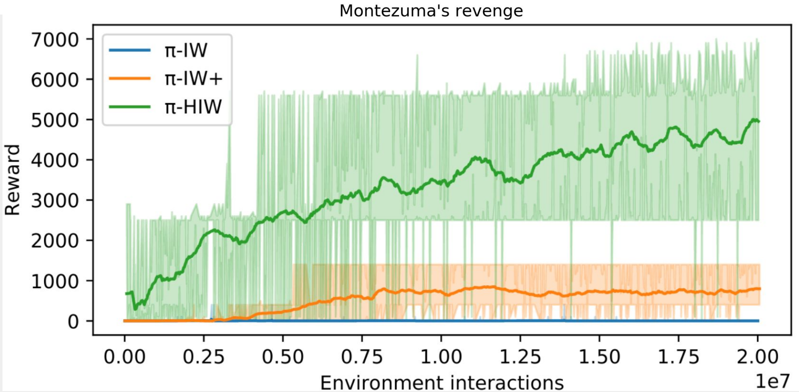


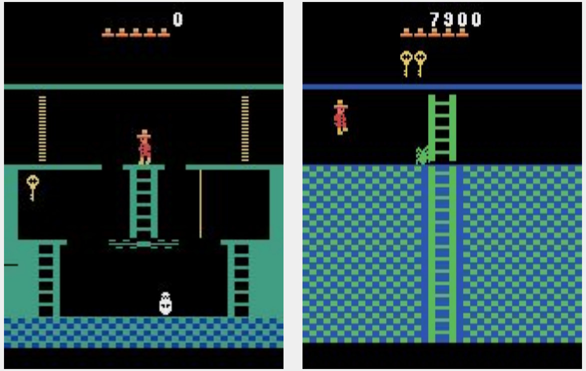




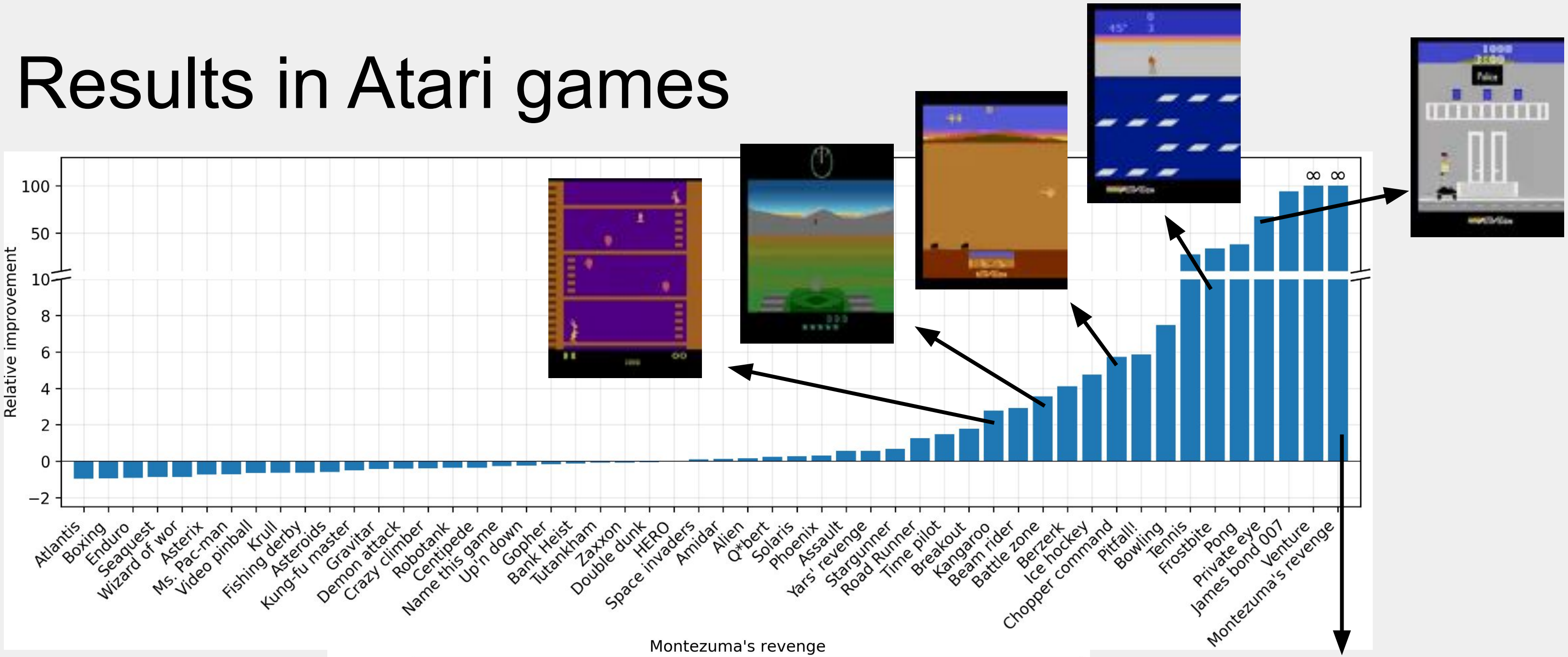
Results in Atari games

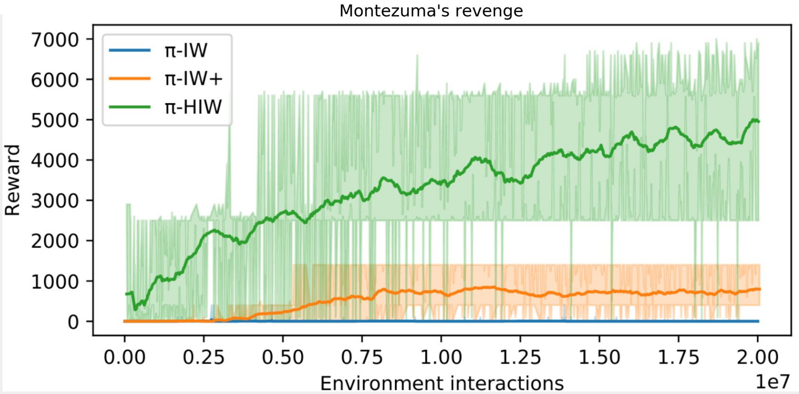


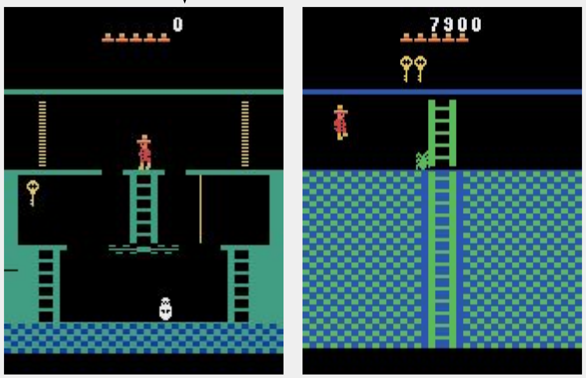




Results in Atari games







Conclusions

- **Tighter bound** for IW(w)
- Simple method for **hierarchical blind search**
- **Hierarchical IW**:
    - Can solve problems of width $w_h + w_\ell$
    - Incremental HIW: **discovering high-level features**
    - Experiments: **IHIW > IW(2)** in 24/36 domains (> instances, < nodes, < time)
- **π-HIW:**
    - **Count-based Rollout IW** (high-level planner)
    - **Improvements to π-IW** (low-level planner):
        - Value
        - Counts
    - Experiments: gridworld and Atari games

Thanks!

https://github.com/aig-upf/hierarchical-IW