Hierarchical Assignment of Behaviours by Self-Organizing

W. Moerman[1] B. Bakker[2] M. Wiering[3]

[1]M.Sc. Cognitive Artificial Intelligence
Utrecht University

[2]Intelligent Autonomous Systems Group
University of Amsterdam

[3]Intelligent Systems Group
Utrecht University

Neural Information Processing Systems 2007 Workshop
Hierarchical Organization of Behaviour: Computational, Psychological and Neural Perspectives

Summary of Ideas

We propose:

- Shifting the design burden:
  - from task decomposition
  - to a suitable abstract representation of the state space

Summary of Ideas

We propose:

- Shifting the design burden:
  - from task decomposition
  - to a suitable abstract representation of the state space
- Using self-organization to figure out which behaviours are needed
  - starting with uncommitted policies
  - learning (parts of) its hierarchical structure
  - no designed or fixed pre/post conditions

Summary of Ideas

We propose:

- Shifting the design burden:
  - from task decomposition
  - to a suitable abstract representation of the state space
- Using self-organization to figure out which behaviours are needed
  - starting with uncommitted policies
  - learning (parts of) its hierarchical structure
  - no designed or fixed pre/post conditions

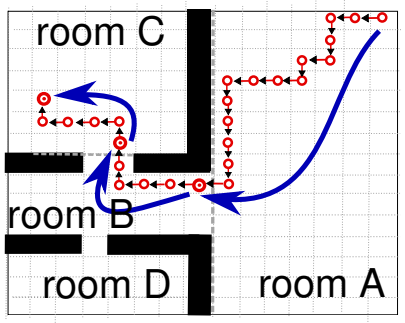
The algorithm is called HABS:
Hierachical **A**ssignment of **B**ehaviours by **S**elf-organization

Outline

1. Introduction
   - Hierarchical Reinforcement Learning
   - Abstractions
2. Our Algorithm (HABS)
   - High Level Policy and Subpolicies
   - Self-Organizing Behaviours
   - Relating HABS to Other Work
3. Experiments
   - Setup
   - Results

Why use hierarchies at all?

Using behaviours (temporally extended/high level actions, . . .) allows

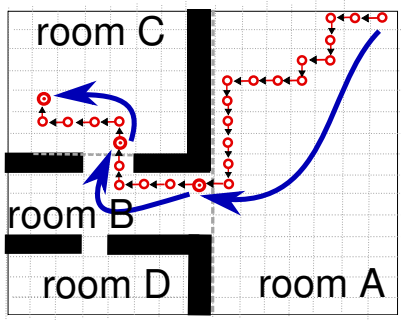


extended actions

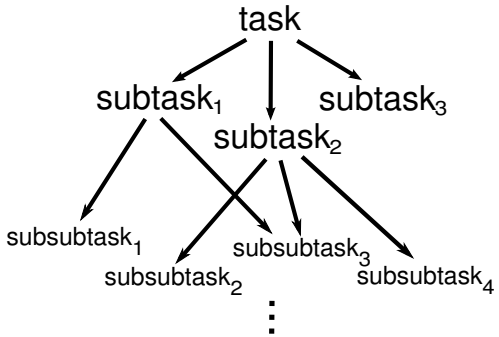
Why use hierarchies at all?

Using behaviours (temporally extended/high level actions, . . .) allows

- "Divide and Conquer": decompose into smaller (easier) subtasks
  - task decomposition enables re-use of (sub)policies



extended actions

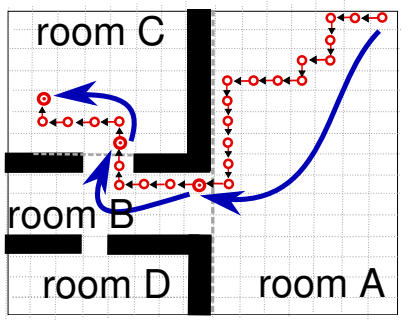


decomposition and re-use

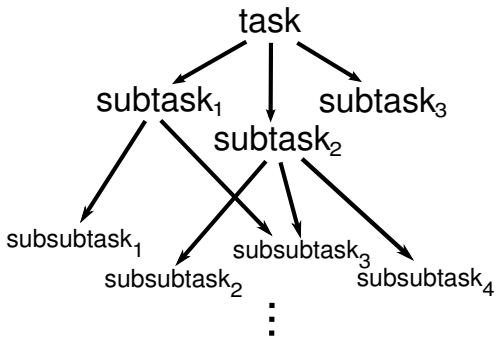
Why use hierarchies at all?

Using behaviours (temporally extended/high level actions, . . .) allows

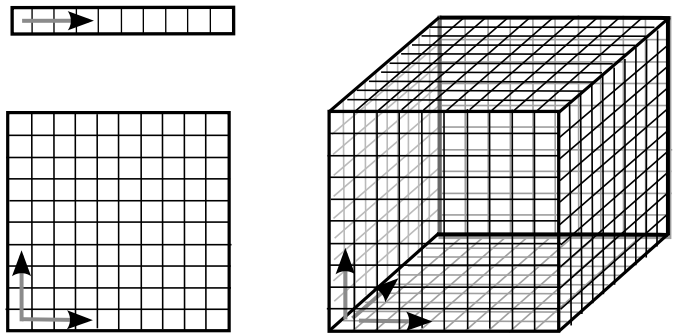
- "Divide and Conquer": decompose into smaller (easier) subtasks
  - task decomposition enables re-use of (sub)policies
- "Dæmon of Dimensionality": smaller state spaces on all levels



extended actions



decomposition and re-use

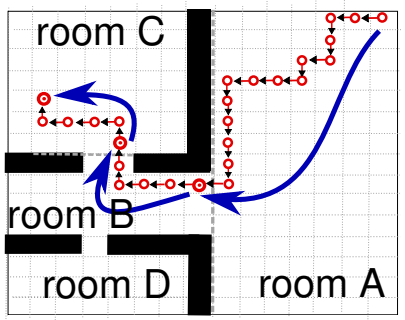


curse of dimensionality

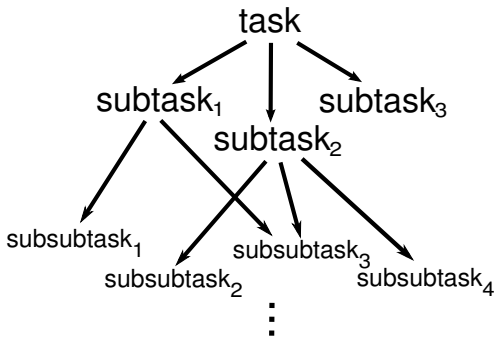
Why use hierarchies at all?

Using behaviours (temporally extended/high level actions, . . .) allows

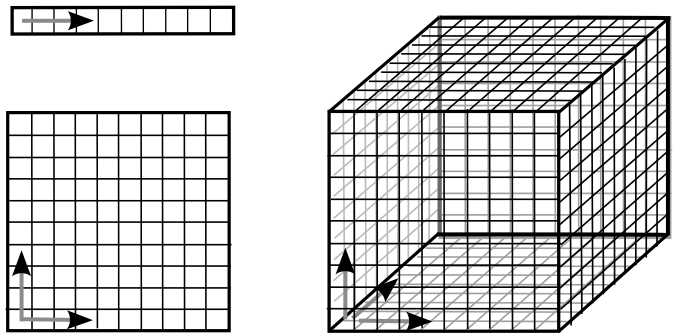
- "Divide and Conquer": decompose into smaller (easier) subtasks
  - task decomposition enables re-use of (sub)policies
- "Dæmon of Dimensionality": smaller state spaces on all levels
- Different specific state abstractions for different (sub)policies (just wait until tomorow)



extended actions



decomposition and re-use

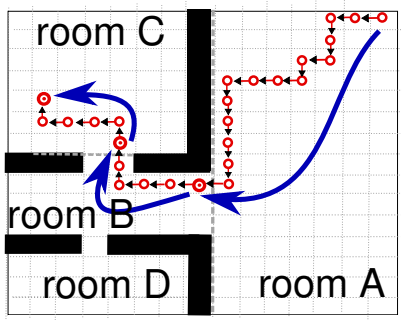


curse of dimensionality

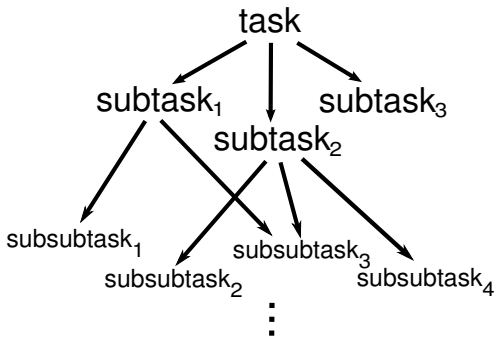
Why use hierarchies at all?

Using behaviours (temporally extended/high level actions, . . .) allows

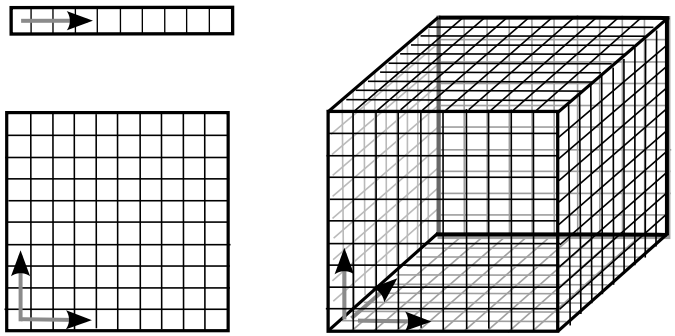
- "Divide and Conquer": decompose into smaller (easier) subtasks
  - task decomposition enables re-use of (sub)policies
- "Dæmon of Dimensionality": smaller state spaces on all levels
- Different specific state abstractions for different (sub)policies (just wait until tomorow)
- Faster exploration



extended actions



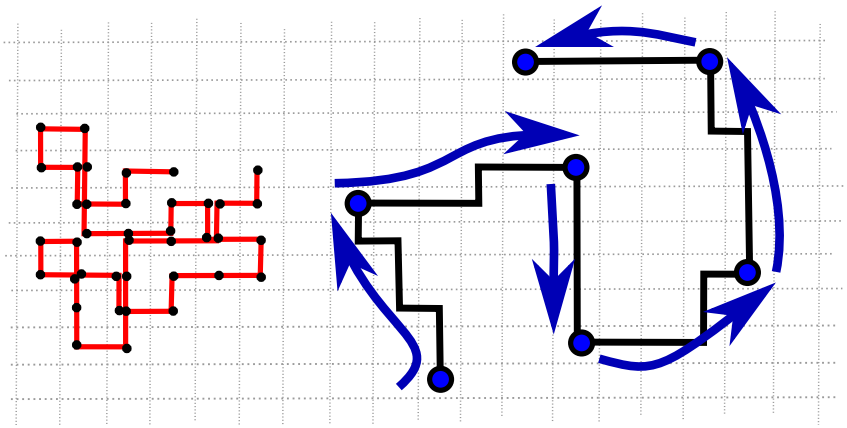
decomposition and re-use



curse of dimensionality

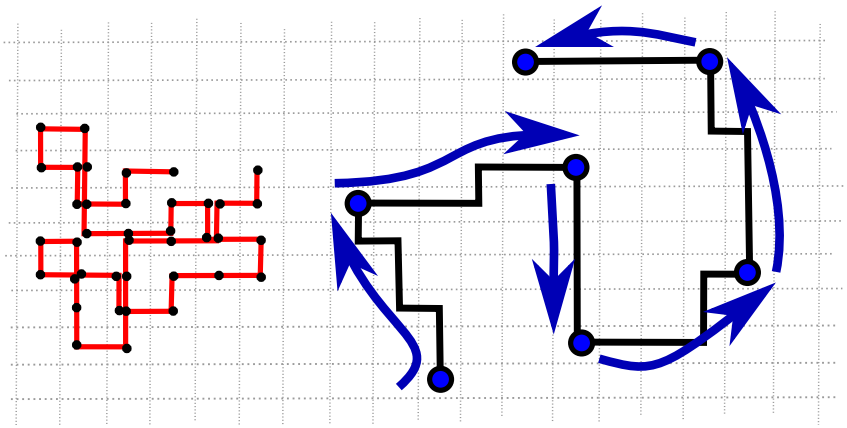
Hierarchies Make Exploration Faster

- Reinforcement Learning exploration is random walk
- but behaviours do something consistent (hopefully)
  - they move agent non-randomly through state space



Hierarchies Make Exploration Faster

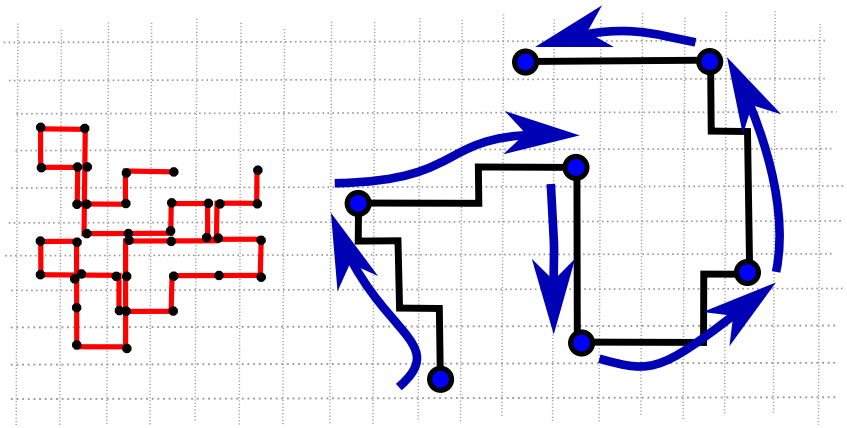
- Reinforcement Learning exploration is random walk
- but behaviours do something consistent (hopefully)
  - they move agent non-randomly through state space



  - less random choices

Hierarchies Make Exploration Faster

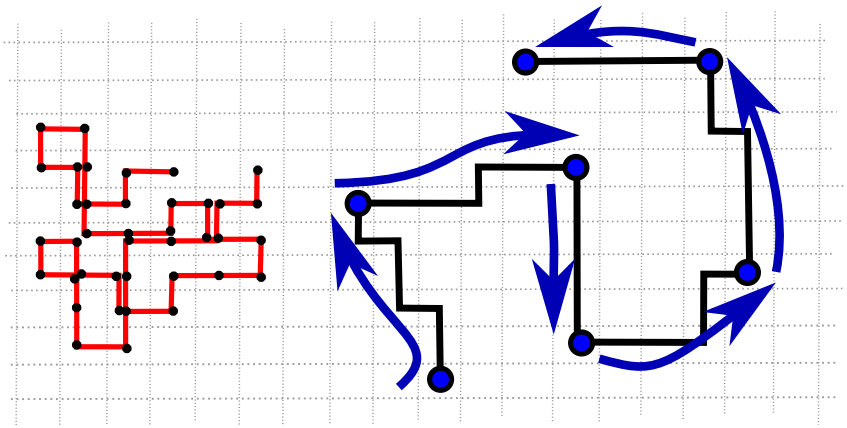
- Reinforcement Learning exploration is random walk
- but behaviours do something consistent (hopefully)
  - they move agent non-randomly through state space



  - less random choices
  - more distance covered in random walk

Hierarchies Make Exploration Faster

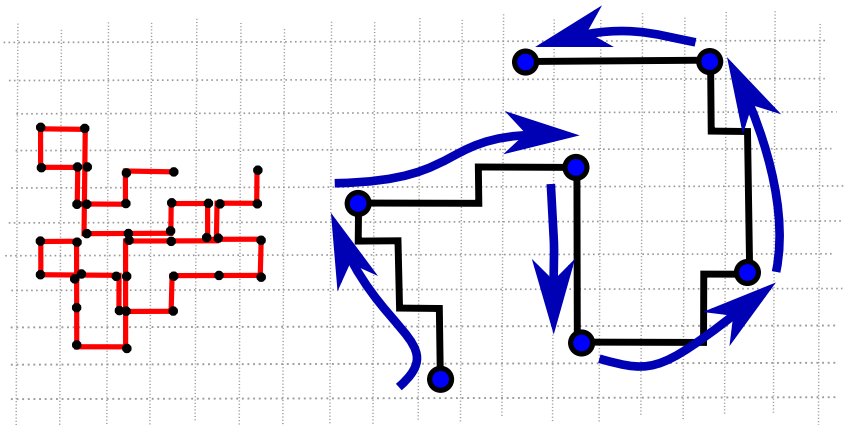
- Reinforcement Learning exploration is random walk
- but behaviours do something consistent (hopefully)
  - they move agent non-randomly through state space



  - less random choices
  - more distance covered in random walk
  - faster exploration

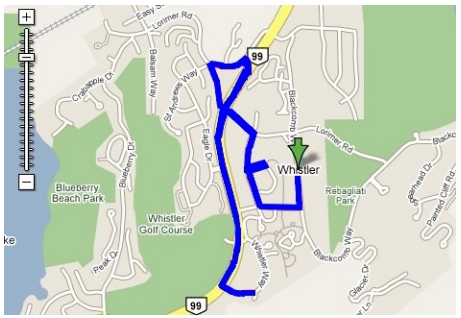
Hierarchies Make Exploration Faster

- Reinforcement Learning exploration is random walk
- but behaviours do something consistent (hopefully)
  - they move agent non-randomly through state space



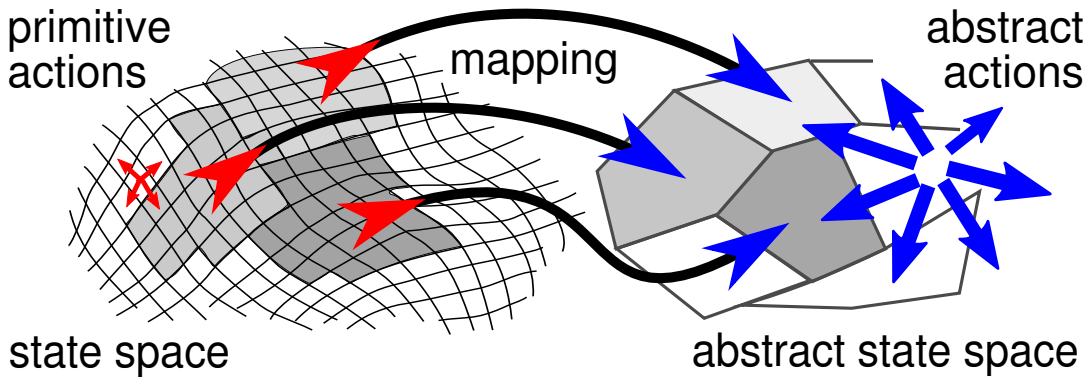
  - less random choices
  - more distance covered in random walk
  - faster exploration

Drunken Mans walk



Random Walks on street level

Abstract State Space

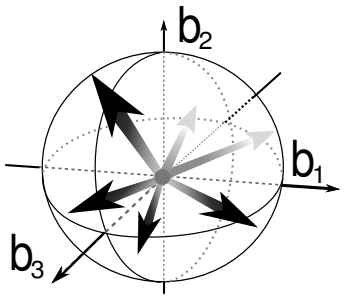


A suitable Abstract State Space has these properties:

- it has an underlying "geometric" structure:
  - not constrained to "spatial" geometry
  - consistent mapping: points close together in state space should be near each other in abstract state space, and vice versa
- Abstract State Space significantly smaller than State Space

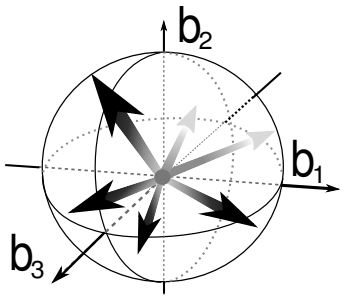
Behaviour Space

- Behaviour Space: space of all possible transition vectors in the Abstract State Space
  - note: Abstract State Space treated as continuous
  - intuition: think of Behaviour Space as a sphere



Behaviour Space

- Behaviour Space: space of all possible transition vectors in the Abstract State Space
  - note: Abstract State Space treated as continuous
  - intuition: think of Behaviour Space as a sphere



(Abstract State Space properties continued . . .)

- Actually occuring transitions between abstract states need to be distributed non-uniformly in the Behaviour Space
  - behaviours (transitions) in abstract state space are vectors in Behaviour Space
  - can be characterized by a limited number of vectors if clustered

Proposed Algorithm

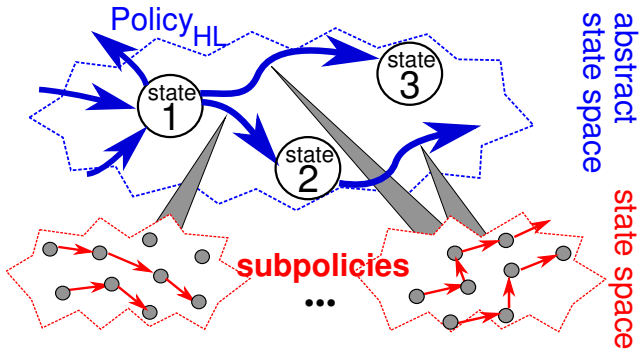
HABS (**H**ierachical **A**ssignment of **B**ehaviours by **S**elf-organization)

- One (high level) $\text{Policy}_{\text{HL}}$ and a limited set of subpolicies
  - uses abstract state space

Proposed Algorithm

HABS (**H**ierachical **A**ssignment of **B**ehaviours by **S**elf-organization)

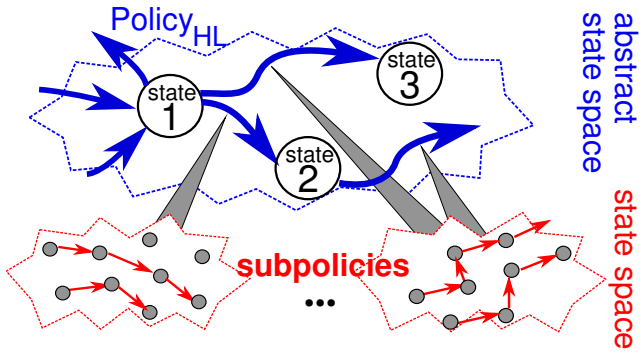
- One (high level) $\text{Policy}_{\text{HL}}$ and a limited set of subpolicies
  - uses abstract state space
- $\text{Policy}_{\text{HL}}$ has subpolicies as its (extended) actions:



Proposed Algorithm

HABS (**H**ierachical **A**ssignment of **B**ehaviours by **S**elf-organization)

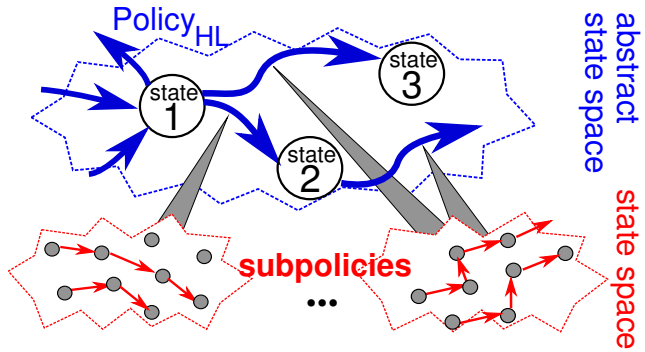
- One (high level) $\text{Policy}_{\text{HL}}$ and a limited set of subpolicies
  - uses abstract state space
- $\text{Policy}_{\text{HL}}$ has subpolicies as its (extended) actions:
  - subpolicies self-organize to cover required behaviours



Proposed Algorithm

HABS (**H**ierachical **A**ssignment of **B**ehaviours by **S**elf-organization)

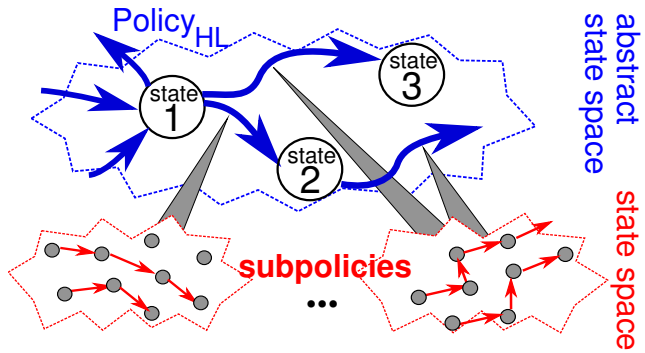
- One (high level) $\text{Policy}_{\text{HL}}$ and a limited set of subpolicies
  - uses abstract state space
- $\text{Policy}_{\text{HL}}$ has subpolicies as its (extended) actions:
  - subpolicies self-organize to cover required behaviours
  - rewards received during a behaviour are accumulated and used for High Level $\text{Policy}_{\text{HL}}$ reward



Proposed Algorithm

HABS (**H**ierachical **A**ssignment of **B**ehaviours by **S**elf-organization)

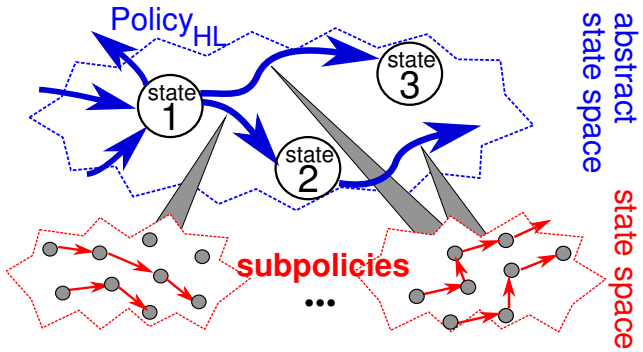
- One (high level) $\text{Policy}_{\text{HL}}$ and a limited set of subpolicies
  - uses abstract state space
- $\text{Policy}_{\text{HL}}$ has subpolicies as its (extended) actions:
  - subpolicies self-organize to cover required behaviours
  - rewards received during a behaviour are accumulated and used for High Level $\text{Policy}_{\text{HL}}$ reward
  - subpolicy rewarding is independent of overall task



Proposed Algorithm

HABS (**H**ierachical **A**ssignment of **B**ehaviours by **S**elf-organization)

- One (high level) $\text{Policy}_{\text{HL}}$ and a limited set of subpolicies
  - uses abstract state space
- $\text{Policy}_{\text{HL}}$ has subpolicies as its (extended) actions:
  - subpolicies self-organize to cover required behaviours
  - rewards received during a behaviour are accumulated and used for High Level $\text{Policy}_{\text{HL}}$ reward
  - subpolicy rewarding is independent of overall task



- Standard RL techniques (online, off policy) like **Q-learning**

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) \quad + \alpha \cdot (reward_t + \gamma \cdot \max_a Q(s_{t+1}, a))$$

Our Solution: Self-Organizing the Behaviours

- Assume: actually occuring behaviours are clustered together
  - use clustering algorithm
  - assign subpolicy to cluster center (specialize)

Our Solution: Self-Organizing the Behaviours

- Assume: actually occuring behaviours are clustered together
  - use clustering algorithm
  - assign subpolicy to cluster center (specialize)
- Exploration: agent will stumble upon new abstract states
  - transitions to new states are behaviours

Our Solution: Self-Organizing the Behaviours

- Assume: actually occuring behaviours are clustered together
  - use clustering algorithm
  - assign subpolicy to cluster center (specialize)
- Exploration: agent will stumble upon new abstract states
  - transitions to new states are behaviours
  - abstract states are nearby

Our Solution: Self-Organizing the Behaviours

- Assume: actually occuring behaviours are clustered together
  - use clustering algorithm
  - assign subpolicy to cluster center (specialize)
- Exploration: agent will stumble upon new abstract states
  - transitions to new states are behaviours
  - abstract states are nearby
  - random walks on small distances are disproportionately better

Our Solution: Self-Organizing the Behaviours

- Assume: actually occuring behaviours are clustered together
  - use clustering algorithm
  - assign subpolicy to cluster center (specialize)
- Exploration: agent will stumble upon new abstract states
  - transitions to new states are behaviours
  - abstract states are nearby
  - random walks on small distances are disproportionately better
  - will happen long before overall task is completed

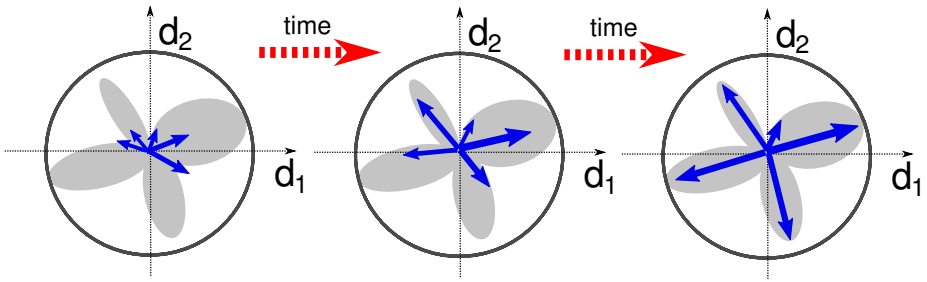
Our Solution: Self-Organizing the Behaviours

- Assume: actually occuring behaviours are clustered together
  - use clustering algorithm
  - assign subpolicy to cluster center (specialize)
- Exploration: agent will stumble upon new abstract states
  - transitions to new states are behaviours
  - abstract states are nearby
  - random walks on small distances are disproportionately better
  - will happen long before overall task is completed

Agent can discover meaningful behaviours long before it has a chance to solve overall problem (due to Random Walks properties)

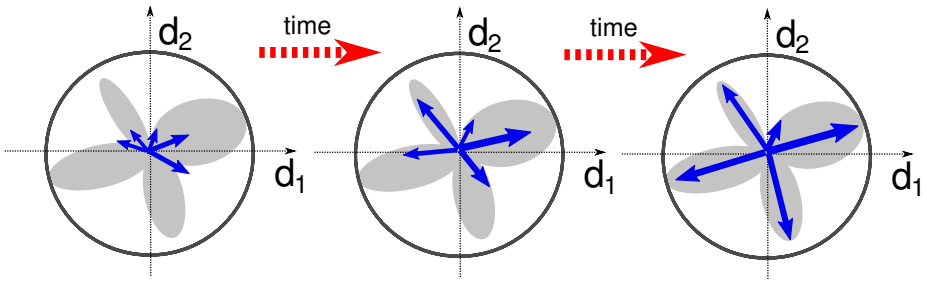
Clustering and Rewarding Subpolicies

- Subpolicy terminates $\Longleftrightarrow$ new abstract state reached or timeout
- On subpolicy termination: compare actually executed behaviour to cluster center (characteristic behaviour) of terminated subpolicy
  - if closest match: move cluster center towards experience



Clustering and Rewarding Subpolicies

- Subpolicy terminates $\Longleftrightarrow$ new abstract state reached or timeout
- On subpolicy termination: compare actually executed behaviour to cluster center (characteristic behaviour) of terminated subpolicy
  - if closest match: move cluster center towards experience



- Always train subpolicy using Reinforcement Learning

$$reward_{sub} = \begin{cases} 0 & \text{not terminated} \\ 1 & \text{terminated: closest match} \\ \kappa_r & \text{terminated: another cluster center is closer} \\ \kappa_f & \text{terminated: timeout (failed to reach anything)} \end{cases}$$

HABS in Pseudo Code

repeat // run HL-Policy

  $Policy_{HL}$ selects *SubPolicy* SUB_i ; // HL-action

  update $Policy_{HL}$ with $reward_{HL}$; // for executing SUB i

until $task\ solved\ or\ timeout_{HL}$

HABS in Pseudo Code

repeat

  $Policy_{HL}$ selects *SubPolicy* SUB_i ;
  repeat // execute SUB i
    SUB_i selects and executes a primitive action ;

    if *new abstract state* **then** BREAK ; // behaviour=>terminate
    else update SUB_i with 0 ; // sparse reward
  until $timeout_{SUB}$

  update $Policy_{HL}$ with $reward_{HL}$;
until *task solved or* $timeout_{HL}$

HABS in Pseudo Code

repeat

  $Policy_{HL}$ selects $SubPolicy\ SUB_i$;

  repeat

    SUB_i selects and executes a primitive action ;

    if *new abstract state* **then** BREAK ;

    else update SUB_i with 0 ;

  until $timeout_{SUB}$

  if $timeout_{SUB}$ **then** punish SUB_i ; `// no new abs. state`

  else `// compare EXECuted with clusters`

    if $EXEC \in CLUSTER_{SUB}$ **then**

      reward SUB_i ; `// match`

      move $CLUSTER_{SUB}$ towards $EXEC$; `// match`

    else punish SUB_i ; `// no match`

  update $Policy_{HL}$ with $reward_{HL}$;

until *task solved or* $timeout_{HL}$

HABS in Pseudo Code

repeat
- $reward_{HL} = 0$; // for accumulating rewards
- $Policy_{HL}$ selects *SubPolicy* SUB_i ;
- **repeat**
  - SUB_i selects and executes a primitive action ;
  - $reward_{HL} \leftarrow reward_{HL} + receivedReward$; // accumulate
  - **if** *new abstract state* **then** BREAK ;
  - **else** update SUB_i with 0 ;
- **until** $timeout_{SUB}$
- **if** $timeout_{SUB}$ **then** punish SUB_i ;
- **else**
  - **if** $EXEC \in CLUSTER_{SUB}$ **then**
    - reward SUB_i ;
    - move $CLUSTER_{SUB}$ towards $EXEC$;
  - **else** punish SUB_i ;
- update $Policy_{HL}$ with $reward_{HL}$;

until *task solved or* $timeout_{HL}$

HABS in Pseudo Code

```
repeat
    reward_HL = 0 ;
    Policy_HL selects SubPolicy SUB_i ;
    repeat
        SUB_i selects and executes a primitive action ;
        reward_HL ← reward_HL + receivedReward ;
        if new abstract state then BREAK ;
        else update SUB_i with 0 ;
    until timeout_SUB
    if timeout_SUB then punish SUB_i ;
    else
        if EXEC ∈ CLUSTER_SUB then
            reward SUB_i ;
            move CLUSTER_SUB towards EXEC ;
        else punish SUB_i ;
    update Policy_HL with reward_HL ;
until task solved or timeout_HL
```

The Difference: Shifting the Burden

HABS differs from other hierarchical RL approaches:

- no focus on defining a task decomposition (MAXQ, HEXQ, HAM)
  - no need to define start and stop conditions
- starts with uncommitted subpolicies that self-organize

HABS shifts design burden from task decomposition to defining a suitable abstract representation

Like many hierarchical approaches, HABS depends on a certain structure (“geometry”) in the State Space

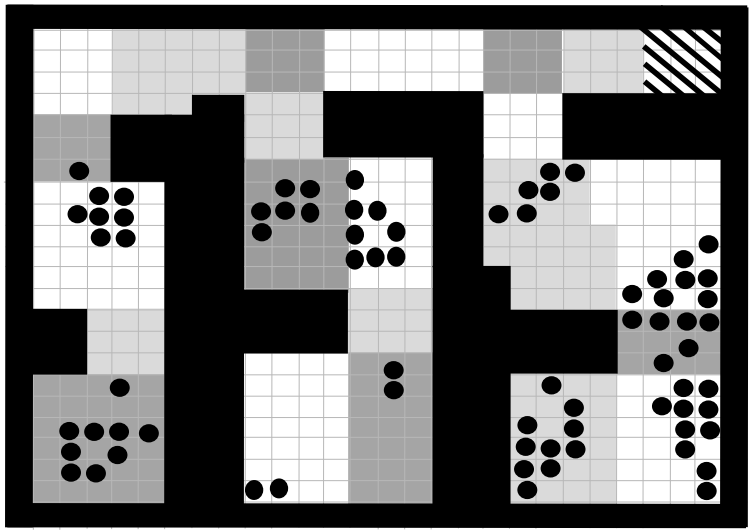
Experiment Description ("Cleaner")

- Gridworld environment:
  - actions: *North, East, South, West, Pickup*$_{object}$, *Drop*$_{object}$
  - walls, drop areas and portable objects (max. 1 per cell)
  - reward for each object dropped at drop area

Experiment Description ("Cleaner")

- Gridworld environment:
  - actions: *North, East, South, West, Pickup*$_{object}$, *Drop*$_{object}$
  - walls, drop areas and portable objects (max. 1 per cell)
  - reward for each object dropped at drop area

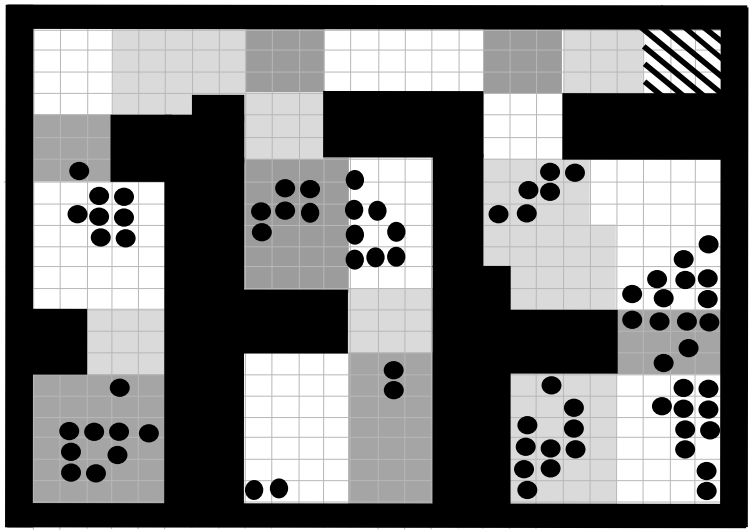
- "Cleaner":
  - many objects
  - agent can carry 10
  - high level policy: multilayer neural network
  - subpolicies: multilayer neural network



Experiment Description ("Cleaner")

- Gridworld environment:
  - actions: *North, East, South, West, Pickup*$_{object}$, *Drop*$_{object}$
  - walls, drop areas and portable objects (max. 1 per cell)
  - reward for each object dropped at drop area

- "Cleaner":
  - many objects
  - agent can carry 10
  - high level policy: multilayer neural network
  - subpolicies: multilayer neural network

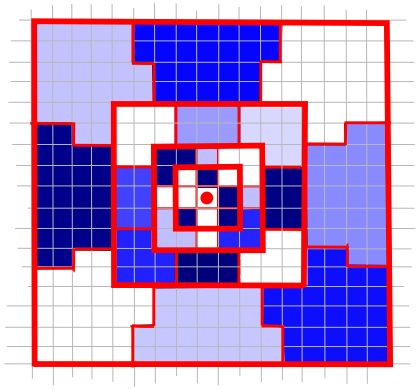


- Task has spatial **and non-spatial** aspects!

State Space and Abstract State Spaces

State Space: (subpolicies)

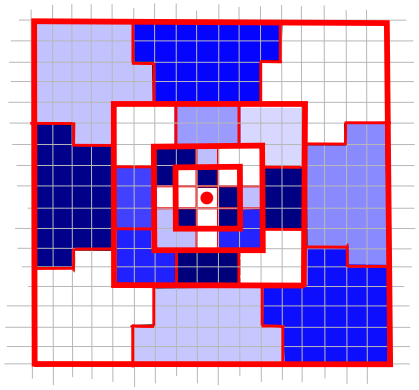
- Agent has a simulated "radar" observes object/wall/drop areas ($\sim$ 100 inputs)



State Space and Abstract State Spaces

State Space: (subpolicies)

- Agent has a simulated "radar" observes object/wall/drop areas ($\sim$ 100 inputs)



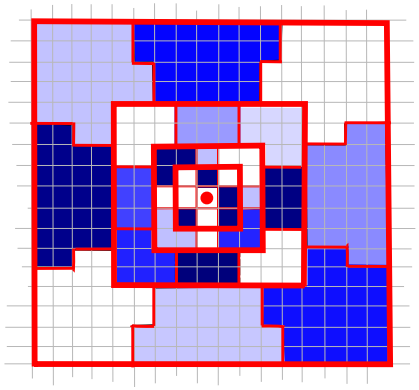
Abstract State Space: (high level policy)

- "cleaner" task uses $< \boldsymbol{area_{agent}}, \boldsymbol{cargo} >$ to determine subpolicy termination
  - no info about other objects!

State Space and Abstract State Spaces

State Space: (subpolicies)

- Agent has a simulated "radar" observes object/wall/drop areas ($\sim$ 100 inputs)



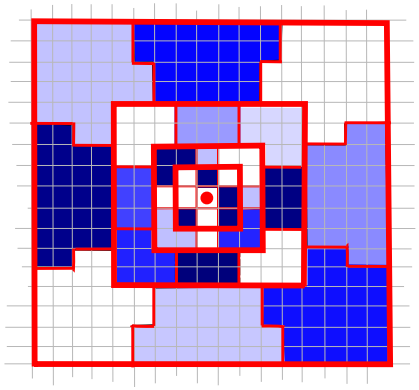
Abstract State Space: (high level policy)

- "cleaner" task uses $< \mathbf{area}_{\mathbf{agent}}, \mathbf{cargo} >$ to determine subpolicy termination
  - no info about other objects!
  - objects moving/(dis)appearing without the agent is no behaviour: agent behaviour only defined by its position and cargo

State Space and Abstract State Spaces

State Space: (subpolicies)

- Agent has a simulated "radar" observes object/wall/drop areas ($\sim$ 100 inputs)



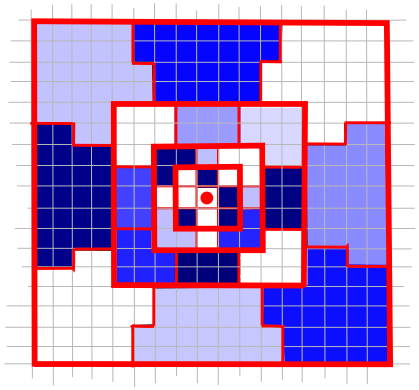
Abstract State Space: (high level policy)

- "cleaner" task uses $< \mathbf{area}_{\mathbf{agent}}, \mathbf{cargo} >$ to determine subpolicy termination
  - no info about other objects!
  - objects moving/(dis)appearing without the agent is no behaviour: agent behaviour only defined by its position and cargo
  - but objects are important for behaviour selection: "high level radar" observations (wider and coarser than low level)

State Space and Abstract State Spaces

State Space: (subpolicies)

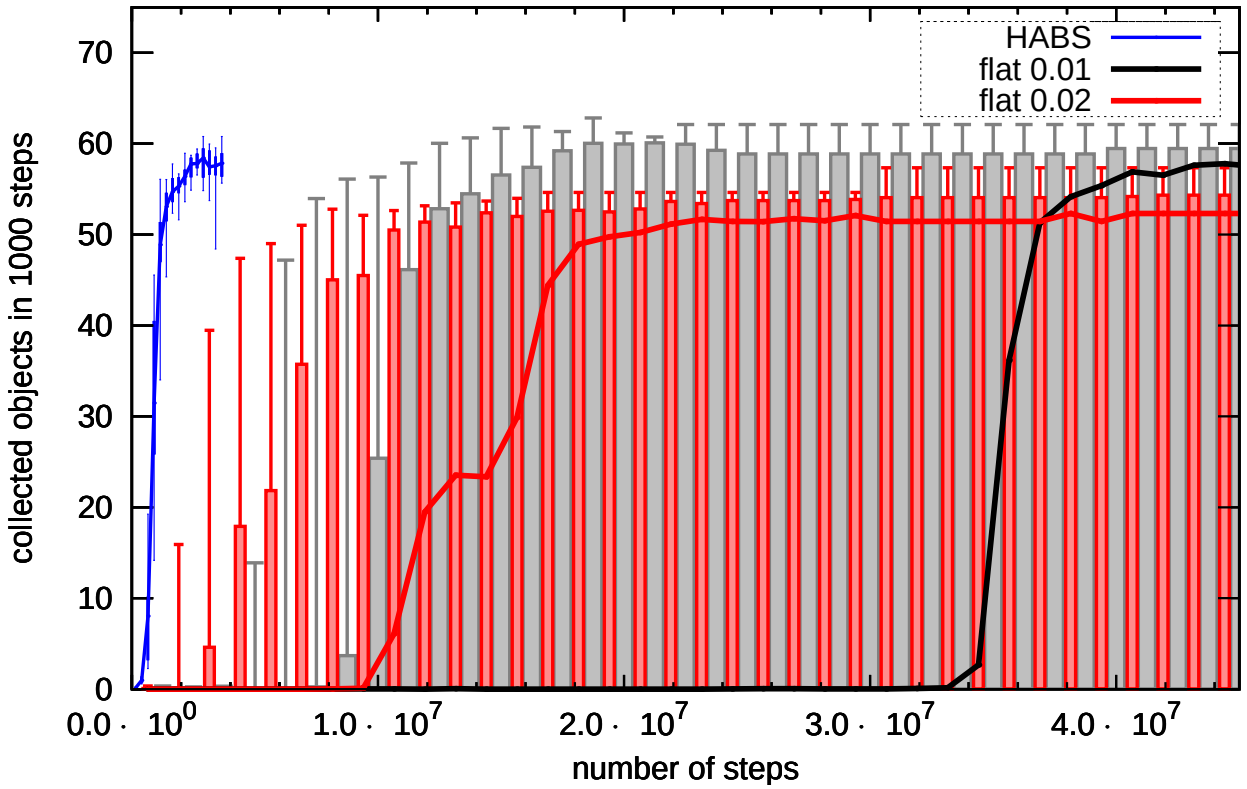
- Agent has a simulated "radar" observes object/wall/drop areas ($\sim$ 100 inputs)



Abstract State Space: (high level policy)

- "cleaner" task uses $< \mathbf{area_{agent}}, \mathbf{cargo} >$ to determine subpolicy termination
  - no info about other objects!
  - objects moving/(dis)appearing without the agent is no behaviour: agent behaviour only defined by its position and cargo
  - but objects are important for behaviour selection: "high level radar" observations (wider and coarser than low level)
- Better to say: "radar" observation **is** Abstract State – and only a subset of the Abstract States is used for subpolicy termination

Cleaner: Results



Boxplots:

- **HABS**

- "flat" learner

$\alpha = \{$**0.02**, **0.01**$\}$

HABS:

5 hidden ($\text{Policy}_{\text{HL}}$)

2 hidden (subpol.)

"flat": 15 hidden

- HABS is much faster and only slightly suboptimal
  - "flat" has wide variance in convergence value ⇕
  - "flat" has far wider variance in convergence time ⟺

HABS Demonstration

- HABS near convergence
- shows suboptimality: agent ignores objects in area I (pink, center)

Conclusions / Future Work

- conclusions about HABS:
  - possible to shift design burden from task decomposition to state space abstraction
  - learns conditions for behaviours by self-organizing
  - can start with unspecified behaviours (no fixed pre/post conditions)

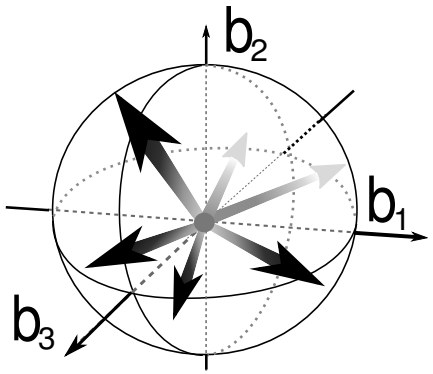
- future work
  - try HABS with 3 or more layers
  - behaviour representation: limited to vectors?


End

Action Space and Primitive Actions

Action Space: space of all possible transition vectors in the state space

- primitive actions are vectors in the *Action Space* (a subset)



- primitive actions are not distributed evenly but clustered together
  - primitive actions are not distributed evenly but clustered together
  - only one primitive action for *North* instead of many $North_1$, $North_2$, $North_3$, ... for going north from *state* 1 to 352, from *state* 2 to 369, from *state* 3 to 792345, ...
- relative (vectors), not absolute (*North = "in state A goto B"*)

back to behaviour space

Behaviours Mirror Primitive Actions

Primitive Actions	Behaviours
vectors in action space	vectors in behaviour space
relative to state	relative to abstract state
clustered	hopefully clustered
1 time step	1 high level time step
action successful	reach new abstract state
action fails	timeout

back to behaviour space

Training Subpolicies, How?

- a subpolicy starts with no knowledge (i.e. randomly initated)
  - what is its desired or characteristic behaviour?
- train on pairs of abstract states?
  - designer needs to specify pre/post conditions
- rewards independent of the overall task ($\text{Policy}_{\text{HL}}$)
  - behaviour "**A** $\Rightarrow$ **goal**" same as "**B** $\Rightarrow$ **C**"
  - **blue behaviour** has high Q_{HL}-value in **A** but low Q_{HL}-value in **B**
  - **red behaviour** has high Q_{HL}-value in **B**

not dependence on $\text{Policy}_{\text{HL}}$ on fixed pre/post conditions!

back to $\text{Policy}_{\text{HL}}$

Reinforcement Learning

An agent:

- observes a state
- executes an action
- receives a reward

Based on this information, it needs to learn what actions to select in what situations – using an RL algorithm:

- future rewards need to be discounted
- stored in tabular form or function approximator

back to intro

Q-Learning and Advantage Learning

Q-Learning:

$$Q(s_t, a_t) \leftarrow (1-\alpha)\cdot Q(s_t, a_t) + \alpha \cdot (reward_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a))$$

Advantage Learning (*Baird*):

$$A(s_t, a_t) \leftarrow (1-\alpha)\cdot A(s_t, a_t) + \alpha \cdot \left(\begin{array}{c} \max_{a_t} A(s_t, a_t) + \\ \frac{reward_{t+1} + \gamma \max_a A(s_{t+1}, a) - \max_{a_t} A(s_t, a_t)}{k} \end{array} \right)$$

where α is the learning rate, and k the scaling factor ($0 < k \leq 1$).
With $k = 1$ this equation reduces to Q-Learning

▸ back to intro ▸ back to Proposed Algorithm ▸ back to Subpolicies

Formula for Clustering

- used *euclidean distance* for determining closest cluster center
- if subpolicy was the winner, move cluster center:

$$char_{t+\Delta t} = (1 - \alpha) \cdot char_t + \alpha \cdot act_{t \to t+\Delta t}$$

  where $act_{t \to t+\Delta t}$ is the actually executed behaviour, Δt the time it took to execute the subpolicy, and $char_t$ the characteristic behaviour vector for a subpolicy at time t
- otherwise: do nothing with the clustering
  - subpolicy executed a behaviour that is better matched by another subpolicy

back to clustering

Details on Self Organizing

- When agent reaches new abstract state it experiences a behaviour
  - center of closest cluster moved towards the newly experienced behaviour
  - this is the clustering part
- characteristic behaviour for a subpolicy is represented by cluster center
  - reward subpolicy when actually executed behaviour closest to its own cluster center
  - forced “outward” by punishing “staying in the same abstract state”
  - the self-organizing part

Formulae for Rewarding Behaviours 1

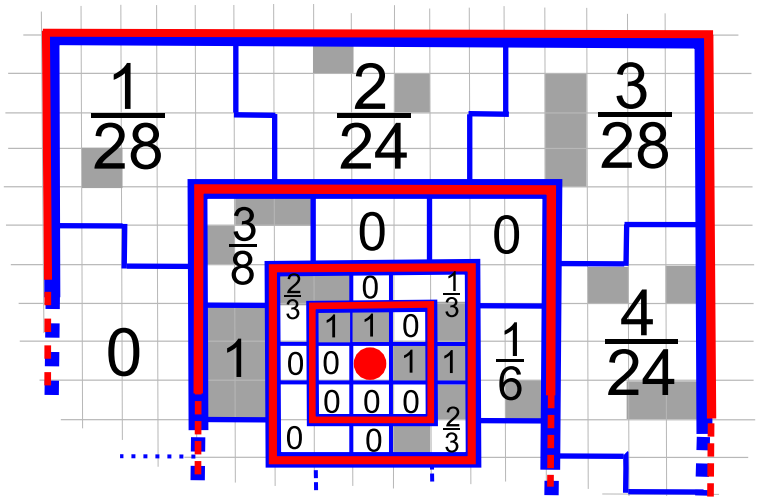
At every time step during the execution of a subpolicy, the normal RL (Q-Learning, Advantage Learning, ...) update is applied:

$$Q_L(s_t, a_t) \leftarrow (1-\alpha) \cdot Q_L(s_t, a_t) + \alpha \cdot (r_{t+1} + \gamma \cdot \max_a Q_L(s_{t+1}, a))$$

Q_L is the Q-Value for a (low level) subpolicy, α and γ are the learning rate and discount as usual

Sensors

- agent has radar-like sensorgrid (8 areas per ring)
  - details nearby, course observations far away
  - trade off between detail and amount of sensor data

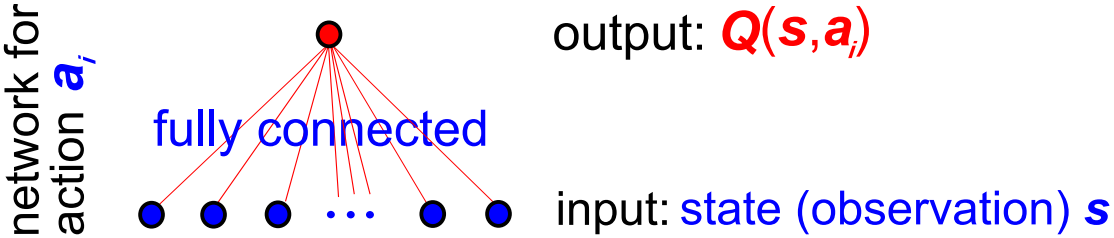


- observation: vector of area densities
  - $\langle \frac{1}{28}, \frac{2}{24}, \frac{3}{28}, \frac{4}{24}, \ldots, \frac{3}{8}, 0, 0, \frac{1}{6}, \ldots, \frac{2}{3}, 0, \frac{1}{3}, 1, \ldots 1, 1, 0, 1, \ldots \rangle$
  - for walls / objects / drop areas, so 3×32 inputs
- extra values coding for position, cargo, etc were added

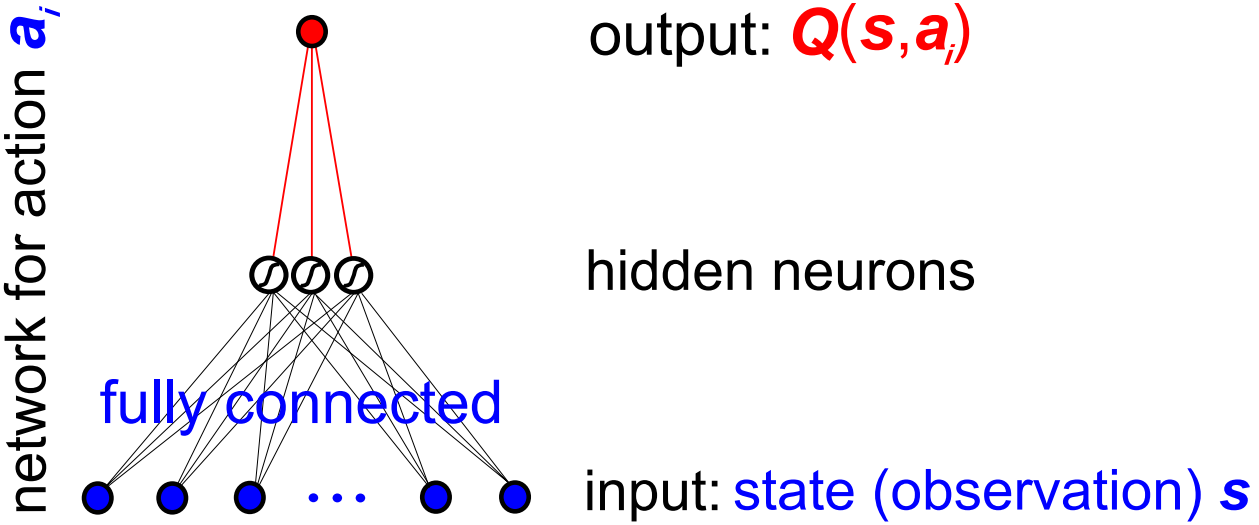
back to state space

Neural Nets for Subpolicies 1

Linear neural network:



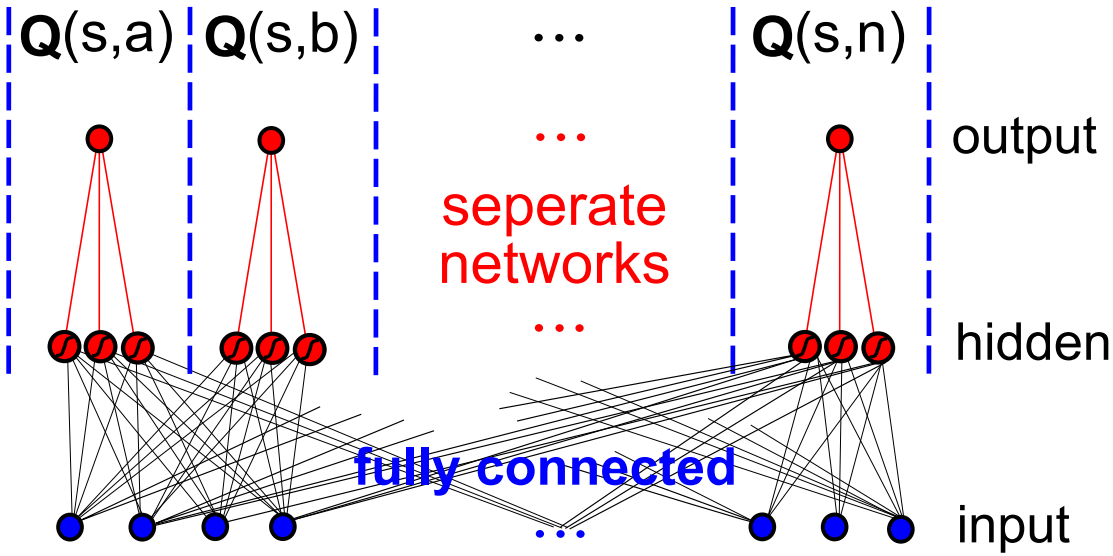
Multilayer Neural network:



Both are “feed forward” networks (no recurrence)

back to experiment description

Neural Nets for Subpolicies 2



- each subpolicy has 6 neural nets, one (shown on prev. slide) for each action a_i, giving $Q(s, a_i)$ in state s
  - allows for different inputs for different actions (not tried here)
- advantage of seperate networks:
  - no interference between actions
  - less hidden neurons needed for each network
  - faster backpropagation (because only one action is updated)

back to experiment description

Comparing with Flat Learners

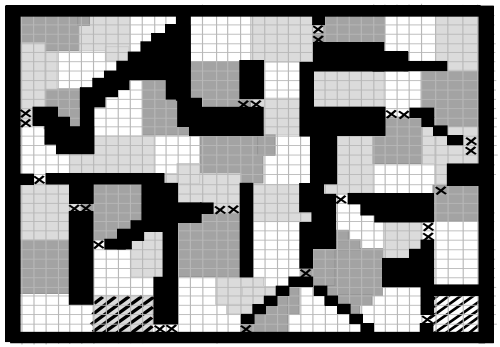
A "flat" reinforcement learning agent was used for comparison

- first experiment (maze): tabular
  - each observation is unique (because position is included)
  - more efficient storage in table (only store position)
- second experiment (cleaner): neural network
  - comparable to what the high level policy used
  - also tried with more hidden neurons

used best performance settings for the "flat" learner (took lots and lots of time to find)

Experiment: Maze

- gridworld environment:
  - actions: *North, East, South, West, Pickup*$_{object}$, *Drop*$_{object}$
  - walls, drop areas and portable objects (max. 1 per cell)
  - reward for each object dropped at drop area

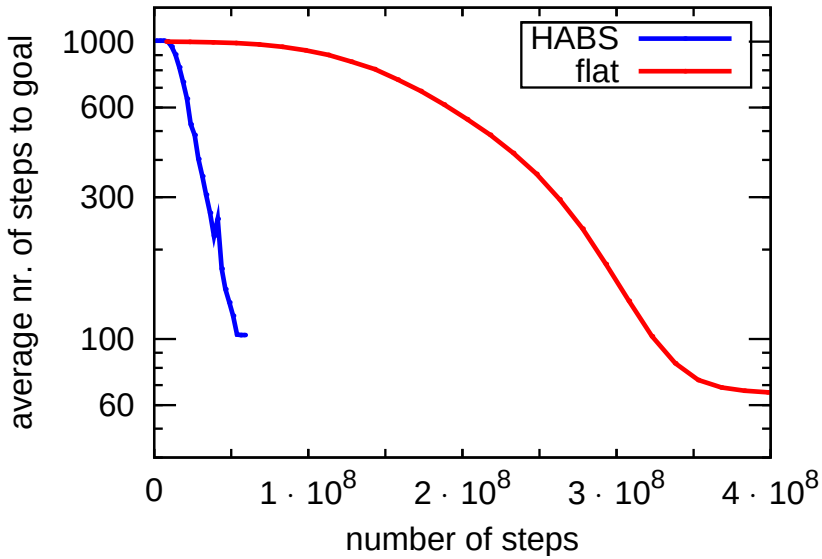


- first experiment (Maze):
  - big ($39 \times 36 \approx 1.4 \cdot 10^3$ cells)
  - only one object
  - high level policy tabular
  - subpolicies: linear neural network

back to Cleaner Description
back to Cleaner Results

First Experiment (Maze): Results

- HABS compared with "flat" learner (best performance, tabular)
- for HABS, only coarse search was done, but no extensive fine tuning!



- HABS is order of magnitude faster but sub optimal
- memory usage:
  - flat learner needs to store 10^7 Q-values ($\approx$ 100 megabyte)
  - HABS needs $5 \cdot 10^3 \times$ *numberOfSubpolicies* ($\approx$ 1 megabyte)
  - Neural network storage is neglectable

▸ back to Cleaner Description ▸ back to Cleaner Results