Hierarchical Assignment of Behaviours by Self-Organizing

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

[1]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:

- A new Hierarchical Reinforcement Learning algorithm:
  - Shifting the focus to state abstraction
  - Using self-organization to learn behaviours

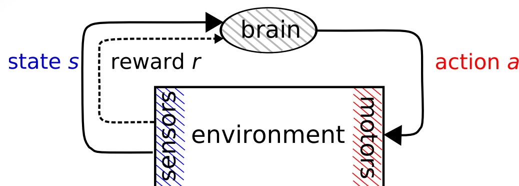
The proposed algorithm is called HABS:
Hierachical **A**ssignment of **B**ehaviours by **S**elf-organization

Outline

1. Introduction
   - Reinforcement Learning
   - Hierarchical Reinforcement Learning
2. Our Algorithm (HABS)
   - Focus on State Abstraction
   - Searching For Suitable Abstractions
   - Self-Organizing the Behaviours
3. Experiments
   - Setup
   - Results

Agent–Environment–Interaction

- Feedback between agent and environment:

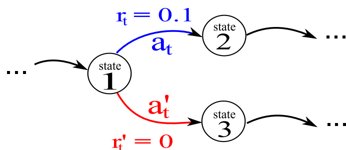


- Agent:
  - **brain** (table or function approximator)
  - sensors
  - motor controls
- Environment:
  - **rewards**
  - states (sensed by sensors)
  - actions (executed by motor controls)

- Agent wants to maximize the total reward it receives

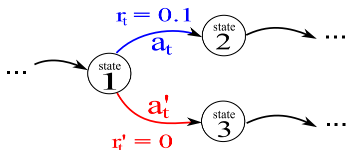
Reinforcement Learning is “Trial–and–Error”

- Suppose we define “goodness” values based on the rewards
  - *being in* a state: $V(state_2) = 0.1$ and $V(state_3) = 0$
  - or: *going to* a state: $Q(state_1, a_t) = 0.1$ and $Q(state_1, a_{t'}) = 0$



Reinforcement Learning is “Trial–and–Error”

- Suppose we define “goodness” values based on the rewards
  - *being in* a state: $V(state_2) = 0.1$ and $V(state_3) = 0$
  - or: *going to* a state: $Q(state_1, a_t) = 0.1$ and $Q(state_1, a_{t'}) = 0$



- Every time the agent visits a state, update the values:
  - move Q or V towards *reward*:

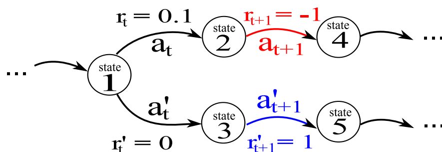
$$V(state) \leftarrow (1-\alpha)\cdot V(state) + \alpha \cdot reward$$

or

$$Q(state, action) \leftarrow (1-\alpha)\cdot Q(state, action) + \alpha \cdot reward$$

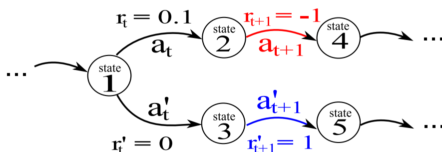
Deceptive Rewards — Discounting the Future

- What if the future rewards are "deceptive"?
  - a good action followed by a very bad action
  - future needs to be taken into account



Deceptive Rewards — Discounting the Future

- What if the future rewards are “deceptive”?
  - a good action followed by a very bad action
  - future needs to be taken into account

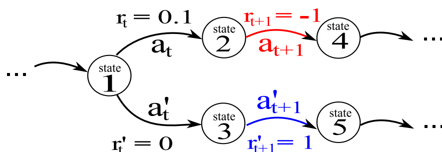


- Some measurement for future is needed: FUTUREGOODNESS

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \left(r_{t+1} + \text{FUTUREGOODNESS}\right)$$

Deceptive Rewards — Discounting the Future

- What if the future rewards are “deceptive”?
  - a good action followed by a very bad action
  - future needs to be taken into account

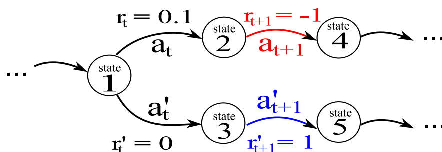


- Some measurement for future is needed: FUTUREGOODNESS
  - SARSA: FUTUREGOODNESS $= \gamma \cdot Q(s_{t+1}, a_{t+1})$

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1})\right)$$

Deceptive Rewards — Discounting the Future

- What if the future rewards are “deceptive”?
  - a good action followed by a very bad action
  - future needs to be taken into account



- Some measurement for future is needed: FUTUREGOODNESS
  - SARSA: FUTUREGOODNESS $= \gamma \cdot Q(s_{t+1}, a_{t+1})$
  - Q-Learning: FUTUREGOODNESS $= \gamma \cdot \max_a Q(s_{t+1}, a)$

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

Why use hierarchies at all?

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

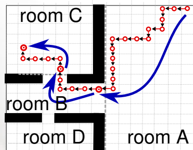
room C
room B
room D
room A

extended actions

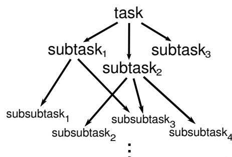
Why use hierarchies at all?

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

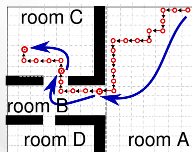


decomposition and re-use

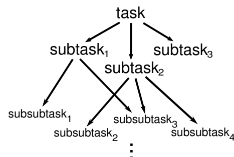
Why use hierarchies at all?

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

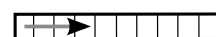
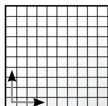
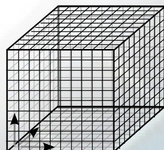
- "Divide and Conquer": decompose into smaller (easier) subtasks
  - task decomposition enables re-use of (sub)policies
- Battling the "Dæmon of Dimensionality"
  - smaller state spaces on all levels



extended actions



decomposition and re-use

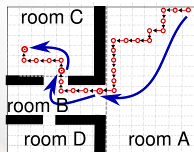




curse of dimensionality

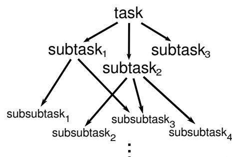
Why use hierarchies at all?

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

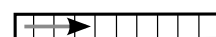
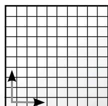
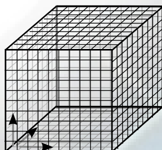
- "Divide and Conquer": decompose into smaller (easier) subtasks
  - task decomposition enables re-use of (sub)policies
- Battling the "Dæmon of Dimensionality"
  - smaller state spaces on all levels
- Faster exploration ≫



extended actions



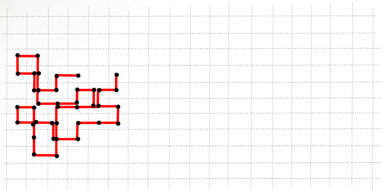
decomposition and re-use

curse of dimensionality

Hierarchies Make Exploration Faster

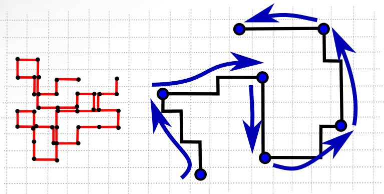
- Reinforcement Learning exploration is random walk
  - a new random action every time step

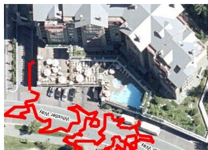


Drunken Mans Walk

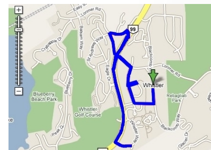
Hierarchies Make Exploration Faster

- Reinforcement Learning ▸ exploration is random walk
  - a new random action every time step





Drunken Mans Walk

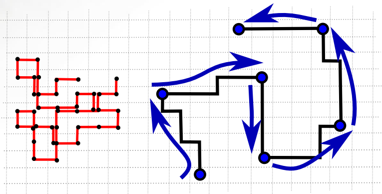


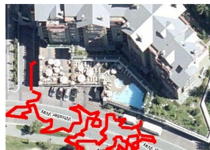
Random Walk on street level

- But behaviours do something consistent (hopefully)
  - they move agent non-randomly through state space
  - Less random choices (only when selecting behaviour)
  - more distance: faster exploration

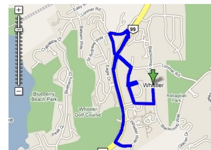
Hierarchies Make Exploration Faster

- Reinforcement Learning ▸ exploration is random walk
  - a new random action every time step





Drunken Mans Walk



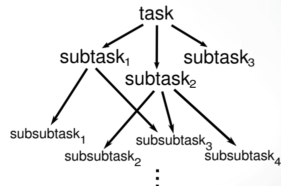
Random Walk on street level

- But behaviours do something consistent (hopefully)
  - they move agent non-randomly through state space
  - Less random choices (only when selecting behaviour)
  - more distance: faster exploration

Agent can discover meaningful behaviours long before it has a chance to solve overall problem

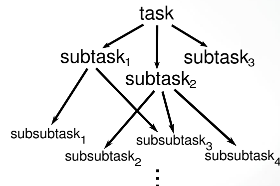
Shifting Focus

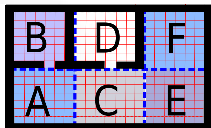
- Many Hierarchical RL-approaches
  focus on Task Decomposition
  - designer decomposes actions into smaller and smaller actions
  - what if the task is complicated or the designer doesn't see the solution?
  - MAX-Q, HAM, HEX-Q, ...



Shifting Focus

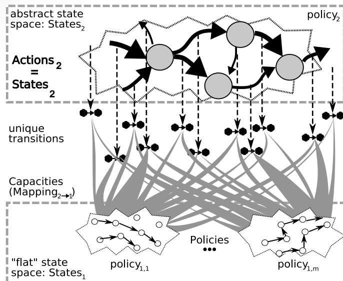
- Many Hierarchical RL-approaches **focus on Task Decomposition**
  - designer decomposes actions into smaller and smaller actions
  - what if the task is complicated or the designer doesn't see the solution?
  - MAX-Q, HAM, HEX-Q, ...



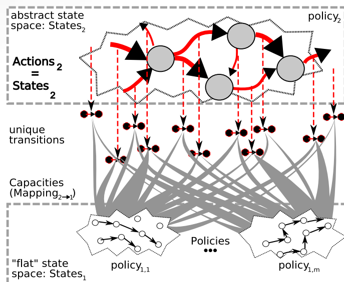


- **Focus on State Space Abstraction**
  - complementary to task decomposition
  - provide suitable state abstraction, and let the RL-agent figure it out!
  - done in FEUDAL LEARNING and HASSLE

Abstract States = High level Actions? (Goal Directed)

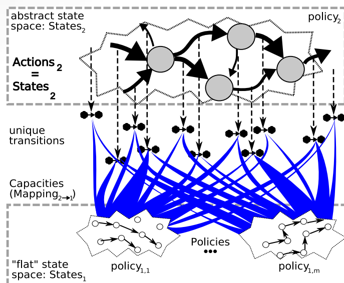


Abstract States = High level Actions? (Goal Directed)



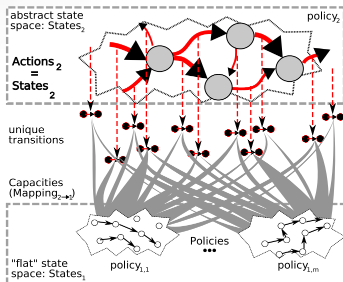
- Each transition (action) between high level states is unique

Abstract States = High level Actions? (Goal Directed)



- Each transition (action) between high level states is unique
- Need to map these to small set of subpolicies

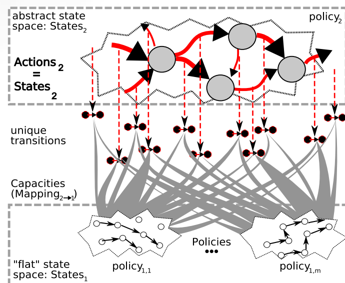
Abstract States = High level Actions? (Goal Directed)

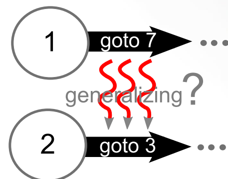


- Each transition (action) between high level states is unique
- Need to map these to small set of subpolicies

- Problem: High level states are **unique**
  - ||high level actions|| = ||high level states|| ∼ "explosion"

Abstract States = High level Actions? (Goal Directed)

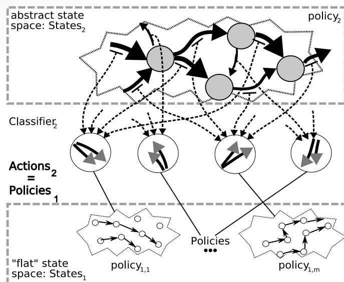




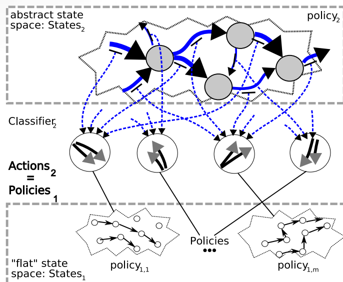
- Each transition (action) between high level states is unique
- Need to map these to small set of subpolicies

- Problem: High level states are **unique**
  - ||high level actions|| = ||high level states|| ∼ "explosion"
  - no generalization or neural networks on high level!
    (no structure in state descriptions)

Abstract States ≠ High level Actions!

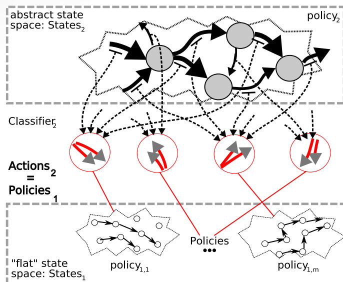


Abstract States ≠ High level Actions!



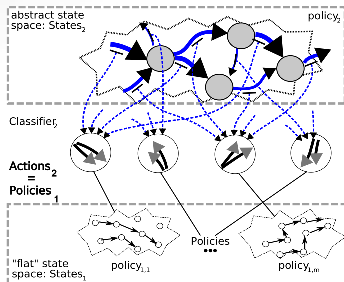
- Reverse order: first many similar transitions are classified

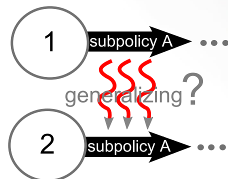
Abstract States ≠ High level Actions!



- Reverse order: first many similar transitions are classified
- Then each cluster corresponds with one subpolicy

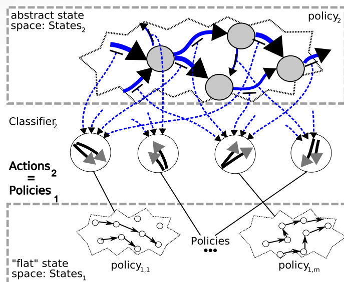
Abstract States ≠ High level Actions!

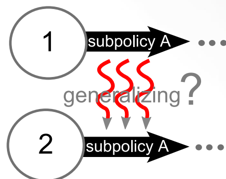




- Reverse order: first many similar transitions are classified
- Then each cluster corresponds with one subpolicy
- Use subpolicy as a high level action (still no generalizing)

Abstract States ≠ High level Actions!



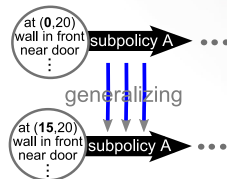


- Reverse order: first many similar transitions are classified
- Then each cluster corresponds with one subpolicy
- Use subpolicy as a high level action (still no generalizing)

- Problem: how dow we find this **classification**?
  - given *a priori* by designer
  - discovered by the agent during learning

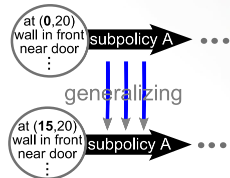
Classification Requires Structure

- Abstract State Space needs structure similar to “flat” state space
  - no structure, no heuristics (e.g. hierarchies)!
  - structure makes generalization possible

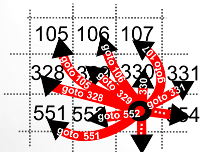


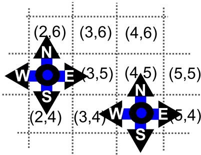
Classification Requires Structure

- Abstract State Space needs structure similar to “flat” state space
  - no structure, no heuristics (e.g. hierarchies)!
  - structure makes generalization possible



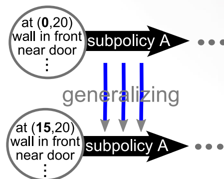
- Primitive actions are not goal directed but relative to a state
  - 1000 cell grid world with only 4 actions!



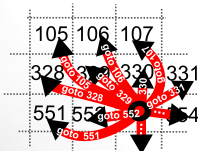


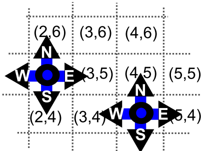
Classification Requires Structure

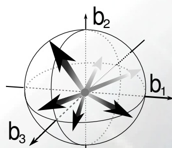
- Abstract State Space needs structure similar to "flat" state space
  - no structure, no heuristics (e.g. hierarchies)!
  - structure makes generalization possible



- Primitive actions are not goal directed but relative to a state
  - 1000 cell grid world with only 4 actions!



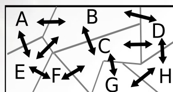




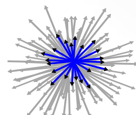
- Action Space:
  - *"space of all possible difference vectors in the state space"*
  - primitive actions all map to few vectors in the Action Space!

Smart State Abstractions

- Suppose we would create this Abstract State Space . . .



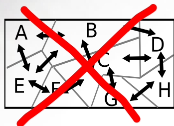
- . . . yielding this Behaviour Space
- no clustering, no natural groups!



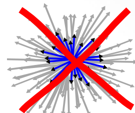
Note: “Behaviour Space” is short for “Abstract Action Space”

Smart State Abstractions

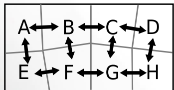
- Suppose we would create this Abstract State Space . . .



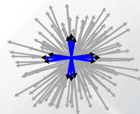
  - . . . yielding this Behaviour Space
  - no clustering, no natural groups!



- Or this . . .

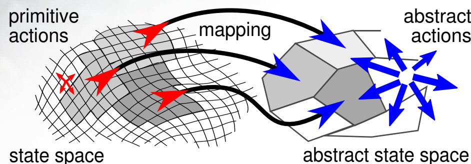


  - . . . yielding this Behaviour Space
  - groups suitable for classification!
  - like the primitive actions



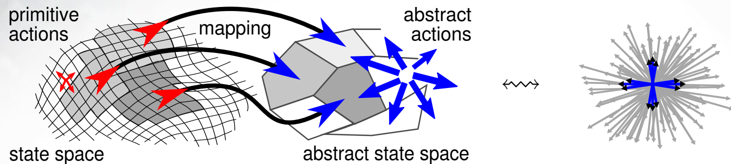
Note: "Behaviour Space" is short for "Abstract Action Space"

Abstract State Space Properties



- 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
  - transitions in abstract state space correspond to meaningful behaviours in “flat” state space
- Abstract State Space significantly smaller than State Space

Abstract State Space Properties



- 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
  - transitions in abstract state space correspond to meaningful behaviours in "flat" state space
- Abstract State Space significantly smaller than State Space
- Actually occurring transitions need to be distributed non-uniformly in Behaviour Space

Proposed Algorithm

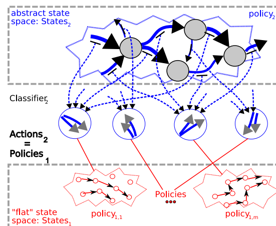
HABS (**H**ierachical **A**ssignment of **B**ehaviours by **S**elf-organization)

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

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

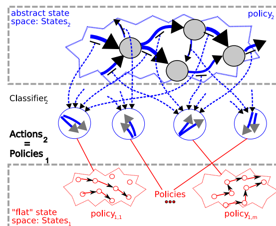


- $\text{Policy}_{\text{HL}}$ has subpolicies as its (extended) actions:
  - the classification and subpolicies are self-organizing
  - rewards received during a behaviour are accumulated and used for High Level $\text{Policy}_{\text{HL}}$ reward
  - subpolicy rewarding is independent of overall task ≫

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



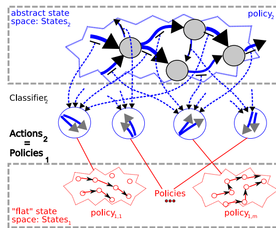
- $\text{Policy}_{\text{HL}}$ has subpolicies as its (extended) actions:
  - the classification and subpolicies are self-organizing
  - rewards received during a behaviour are accumulated and used for High Level $\text{Policy}_{\text{HL}}$ reward
  - subpolicy rewarding is independent of overall task ≫

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

Proposed Algorithm

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

- One (high level) Policy_{HL} and a limited set of subpolicies
  - uses abstract state space



- Policy_{HL} has subpolicies as its (extended) actions:
  - the classification and subpolicies are self-organizing
  - rewards received during a behaviour are accumulated and used for High Level Policy_{HL} reward
  - subpolicy rewarding is independent of overall task ≫

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

- First solve subtasks (subpolicies), then overall task (Policy_{HL})

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

Training Subpolicies, How?

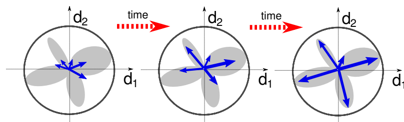
- a subpolicy starts with no knowledge (i.e. randomly initated)
  - what is its desired or characteristic behaviour?
- train on pairs of abstract states?
  - designer needs to specify pre/post conditions

- rewards independent of the overall task ($\text{Policy}_{\text{HL}}$)
  - behaviour **"A ⇒ goal"** same as **"B ⇒ 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**

no dependence on $\text{Policy}_{\text{HL}}$ on fixed pre/post conditions!

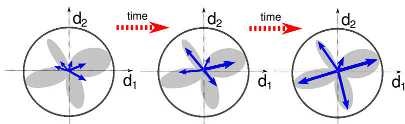
Clustering and Rewarding Subpolicies

- 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 ∈ 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 ← 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 ∈ 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
```

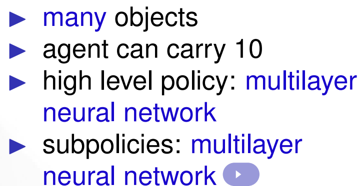
Experiment Description ("Cleaner")

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

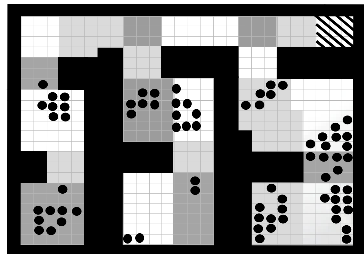
Task has spatial **and non-spatial** aspects!

Experiment Description ("Cleaner")

- Gridworld environment:
  - actions: North, East, South, West, Pickup_{object}, Drop_{object}
  - walls, drop areas and portable objects (max. 1 per cell)
  - rewarded for delivered objects (at drop areas)

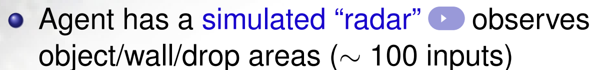


- "Cleaner":
  - many objects
  - agent can carry 10
  - high level policy: multilayer neural network
  - subpolicies: multilayer neural network



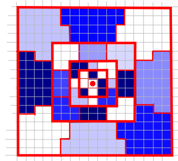
Task has spatial **and non-spatial** aspects!

State Space and Abstract State Spaces



State Space: (subpolicies)

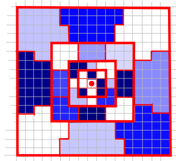
- Agent has a simulated "radar" observes object/wall/drop areas ($\sim$ 100 inputs)



State Space and Abstract State Spaces

State Space: (subpolicies)

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



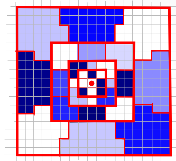
Abstract State Space: ($\text{Policy}_{\text{HL}}$)

- “cleaner” task only uses $< \textbf{\textit{area}}_{\textbf{\textit{agent}}}, \textbf{\textit{cargo}} >$ 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)



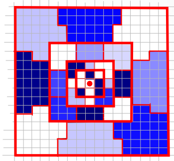
Abstract State Space: ($\text{Policy}_{\text{HL}}$)

- “cleaner” task only uses $< \boldsymbol{area_{agent}}, \boldsymbol{cargo} >$ to determine subpolicy termination
  - no info about other objects!
  - objects moving/(dis)appearing on themselves 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)

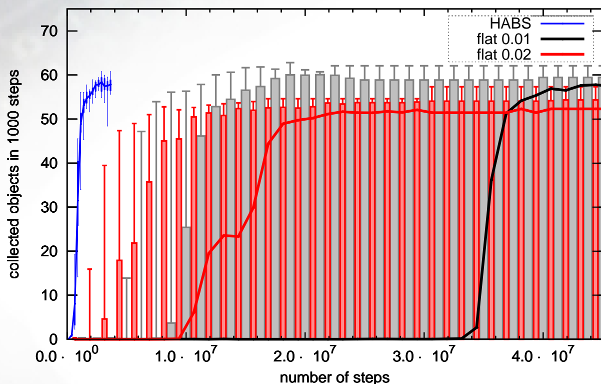


Abstract State Space: (Policy$_{HL}$)

- “cleaner” task only uses $<$ ***area$_{agent}$***, ***cargo*** $>$ to determine subpolicy termination
  - no info about other objects!
  - objects moving/(dis)appearing on themselves 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 – shows that termination criteria can differ from abstract states

Cleaner: Results



Boxplots:

- **HABS**

- "flat" learner

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

HABS:

5 hidden (Policy_{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 ⟺

Demonstration

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

Conclusions / Future Work

- Conclusions about HABS:
  - shifting design burden from task decomposition to state space abstraction is possible
  - learns conditions for behaviours by self-organizing
  - starts with unspecified behaviours (no fixed pre/post conditions)

Conclusions / Future Work

- Conclusions about HABS:
  - shifting design burden from task decomposition to state space abstraction is possible
  - learns conditions for behaviours by self-organizing
  - starts with unspecified behaviours (no fixed pre/post conditions)

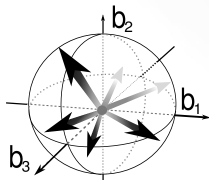
- Future work
  - try HABS with 3 or more layers (possible, in theory)
  - behaviour representation: limited to vectors?
  - automatically detect relevant features in abstract state space which determine behaviour (instead of selecting them *a priori*)
  - test continuous termination criteria ("distance traveled", instead of "reached new abstract state")

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 $North_1, North_2, North_3, \ldots$ 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**

no 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*):

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

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:

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

  where $act_{t \rightarrow t+\Delta t}$ 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

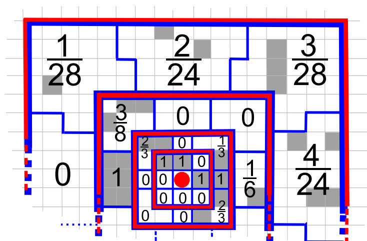
At every time step during the execution of a subpolicy, the normal RL (Q-Learning, Advantage Learning, ...) update is applied:

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

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

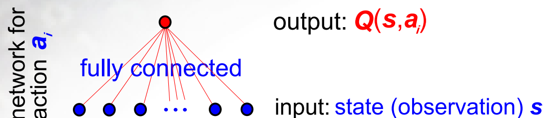


- observation: vector of area densities
  - $\langle \frac{1}{28}, \frac{2}{24}, \frac{3}{28}, \frac{4}{24}, \ldots, \frac{3}{8}, 0, 0, \frac{1}{6}, \ldots, \frac{2}{3}, 0, \frac{1}{3}, 1, \ldots 1, 1, 0, 1, \ldots \rangle$
  - for walls / objects / drop areas, so 3×32 inputs
- extra values coding for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

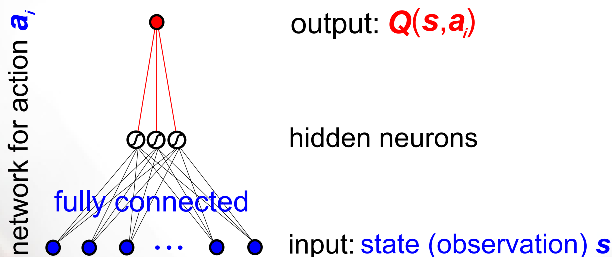
back to state space

Neural Nets for Subpolicies 1

Linear neural network:



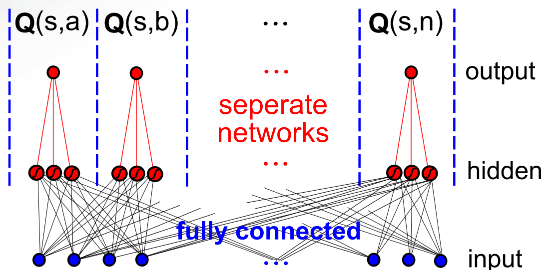
Multilayer Neural network:



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

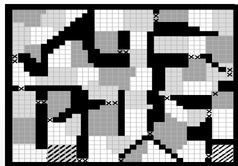
A “flat” reinforcement learning agent was used for comparison

- 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

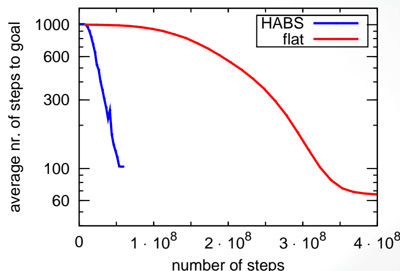


- first experiment (Maze):
  - big ($39 \times 36 \approx 1.4 \cdot 10^3$ cells)
  - only one object
  - high level policy tabular
  - subpolicies: linear neural network

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