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:
**Hierarchical Assignment of Behaviours by Self-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
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
Why use hierarchies at all?

Using **behaviours** (temporarily 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 tomorrow)

---

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 tomorrow)
- **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
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

Random Walks on street level

Drunken Mans walk

Moerman, Bakker, Wiering (UU, UvA) 
Self-Organizing Hierarchical Behaviours
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 occurring 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  (Hierarchical Assignment of Behaviours by Self-organization)

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

HABS (Hierarchical Assignment of Behaviours by Self-organization)

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

HABS (Hierarchical Assignment of Behaviours by Self-organization)

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

HABS (Hierarchical Assignment of Behaviours by Self-organization)

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

**HABS** *(Hierarchical Assignment of Behaviours by Self-organization)*

- One (high level) Policy\(_{HL}\) and a limited set of subpolicies
  - uses abstract state space
- \(\text{Policy}_{HL}\) 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 Policy\(_{HL}\) reward
  - subpolicy rewarding is **independent** of overall task
Proposed Algorithm

HABS (Hierarchical Assignment of Behaviours by Self-organization)

- One (high level) Policy $\text{Policy}_{\text{HL}}$ 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 Policy $\text{Policy}_{\text{HL}}$ reward
  - subpolicy rewarding is independent of overall task

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

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (\text{reward}_t + \gamma \cdot \max_a Q(s_{t+1}, a))$$
Our Solution: Self-Organizing the Behaviours

Assume: actually occurring behaviours are clustered together

- use clustering algorithm
- assign subpolicy to cluster center (specialize)
Our Solution: Self-Organizing the Behaviours

- Assume: actually occurring 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 occurring 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

Moerman, Bakker, Wiering (UU, UvA)
Our Solution: Self-Organizing the Behaviours

Assume: actually occurring 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 occurring 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 occurring 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 ⇔ 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 ▶️

Moerman, Bakker, Wiering (UU, UvA)
Self-Organizing Hierarchical Behaviours
NIPS 2007
Clustering and Rewarding Subpolicies

- Subpolicy terminates $\iff$ 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

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

Always train subpolicy using Reinforcement Learning
Our Algorithm (HABS)

Self-Organizing Behaviours

HABS in Pseudo Code

repeat
  // run HL-Policy
  // HL-action
  $Policy_{HL}$ selects $SubPolicy \; SUB_i$ ;
  // for executing $SUB_i$
  until task solved or $timeout_{HL}$

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

Moerman, Bakker, Wiering (UU, UvA)
repeat

\[ Policy_{HL} \text{ selects } SubPolicy \ SUB_i ; \]
repeat
\[ SUB_i \text{ selects and executes a primitive action ;} \]
if \text{new abstract state} then BREAK ; // behaviour=>terminate
else update \ SUB_i \text{ with 0 ;} // sparse reward
until \ timeout_{SUB} \]

update \ Policy_{HL} \text{ with } \ reward_{HL} ;
until task solved or \ timeout_{HL}
Our Algorithm (HABS)

Self-Organizing Behaviours

HABS in Pseudo Code

repeat

\[ Policy_{HL} \text{ selects } SubPolicy \ SUB_i ; \]

repeat

\[ SUB_i \text{ selects and executes a primitive action} ; \]

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

\[ \text{reward } SUB_i ; \] // match
\[ \text{move } CLUSTER_{SUB} \text{ towards } EXEC ; \] // match

else punish \( SUB_i \) ; // no match

update \( Policy_{HL} \) with \( reward_{HL} \) ;

until task solved or \( timeout_{HL} \)
Our Algorithm (HABS)

Self-Organizing Behaviours

HABS in Pseudo Code

repeat
  \( \text{reward}_{HL} = 0 ; \) \hspace{1cm} // for accumulating rewards
  \( Policy_{HL} \) selects \( SubPolicy \) \( SUB_i \) ;
  repeat
    \( SUB_i \) selects and executes a primitive action ;
    \( \text{reward}_{HL} \leftarrow \text{reward}_{HL} + \text{receivedReward} \) ; \hspace{1cm} // 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} \leftarrow 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 \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}
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:
  - 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*\textsubscript{object}, *Drop*\textsubscript{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 (∼ 100 inputs)
State Space and Abstract State Spaces

State Space: (subpolicies)
- Agent has a simulated “radar” observes object/wall/drop areas (∼ 100 inputs)

Abstract State Space: (high level policy)
- “cleaner” task uses < area$_{agent}$, cargo > to determine subpolicy termination
  - no info about other objects!
Experiments

State Space and Abstract State Spaces

State Space: (subpolicies)
- Agent has a simulated “radar” observes object/wall/drop areas (∼ 100 inputs)

Abstract State Space: (high level policy)
- “cleaner” task uses $< \text{area}_{agent}, \text{cargo} >$ 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 (∼ 100 inputs)

Abstract State Space: (high level policy)
- “cleaner” task uses \(< \text{area}_{\text{agent}}, \text{cargo} >\) 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 (∼ 100 inputs)

**Abstract State Space:** (high level policy)

- “cleaner” task uses $<\text{area}_{\text{agent}}, \text{cargo}>$ 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 (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

Moerman, Bakker, Wiering (UU, UvA)
HABS Demonstration

- HABS near convergence
- shows suboptimality: agent ignores objects in area l (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?
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”*)
### Behaviours Mirror Primitive Actions

<table>
<thead>
<tr>
<th>Primitive Actions</th>
<th>Behaviours</th>
</tr>
</thead>
<tbody>
<tr>
<td>vectors in action space relative to state clustered</td>
<td>vectors in behaviour space relative to abstract state hopefully clustered</td>
</tr>
<tr>
<td>1 time step action successful</td>
<td>1 high level time step reach new abstract state</td>
</tr>
<tr>
<td>action fails</td>
<td>timeout</td>
</tr>
</tbody>
</table>

- back to behaviour space
Training Subpolicies, How?

- a subpolicy starts with no knowledge (i.e. randomly initiated)
  - 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 ($\text{Policy}_{HL}$)
  - behaviour “$A \Rightarrow \text{goal}$” 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 $\text{Policy}_{HL}$ on fixed pre/post conditions!
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
Q-Learning and Advantage Learning

Q-Learning:

\[
Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (\text{reward}_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a))
\]

Advantage Learning (Baird):

\[
A(s_t, a_t) \leftarrow (1 - \alpha) \cdot A(s_t, a_t) + \alpha \cdot \left( \max_a A(s_t, a_t) + \frac{\text{reward}_{t+1} + \gamma \max_a A(s_{t+1}, a) - \max_a A(s_t, a)}{k} \right)
\]

where \(\alpha\) is the learning rate, and \(k\) the scaling factor (0 < \(k\) ≤ 1). With \(k = 1\) this equation reduces to Q-Learning.
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, \(\Delta t\) 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
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, \(\alpha\) and \(\gamma\) are the learning rate and discount as usual.
Sensors

- agent has radar-like sensor grid (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 \( \times \) 32 inputs

- extra values coding for position, cargo, etc were added
Neural Nets for Subpolicies 1

Linear neural network:
- output: $Q(s,a)$
- input: state (observation) $s$

Multilayer Neural network:
- output: $Q(s,a)$
- input: state (observation) $s$

Both are “feed forward” networks (no recurrence)
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 separate networks:
  - no interference between actions
  - less hidden neurons needed for each network
  - faster backpropagation (because only one action is updated)
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\text{object}, Drop\text{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
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