Hierarchical Reinforcement Learning

Thomas G. Dietterich
Department of Computer Science
Oregon State University, Corvallis, Oregon 97331
tgd@cs.orst.edu
http://www.cs.orst.edu/~tgd

On sabbbatical leave:
Thomas G. Dietterich, Visiting Senior Scientist
Institut D'Investigació en Intel.ligéncia Artificial
Consejo Superior de Investigaciones Científicas
Campus de la UAB; 08193-Bellaterra, Barcelona, España

Supported by AFOSR grant F49620-98-1-0375 and a grant from the Spanish Scientific Research Council

Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
  - Defining Subcomponents (Tasks; Macros)
  - Learning Problems and Learning Algorithms
  - Dealing with Suboptimality
  - State Abstraction
  - Discovering Subtasks
  - Open Problems
- **Multi-Agent Decomposition**
  - Defining Subtasks
  - Solution Methods
  - Open Problems and Speculations
- **Review of Key Points**

Introduction

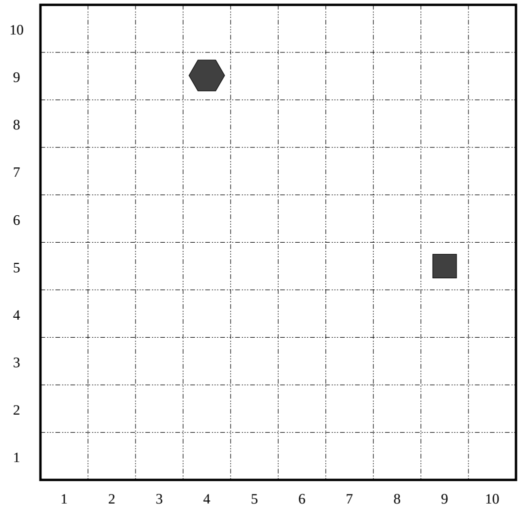
Purpose of Hierarchical RL:

- **Scale-up:** Decompose large problems into smaller ones
- **Transfer:** Share/Re-use Subtasks
- **Overcome Partial Observability**

Key Ideas:

- **Impose constraints on the value function/policy**
  e.g., by identifying structure in the problem
- **Develop learning algorithms that exploit these constraints**

Example 1: Hierarchical Distance to Goal (Kaelbling)

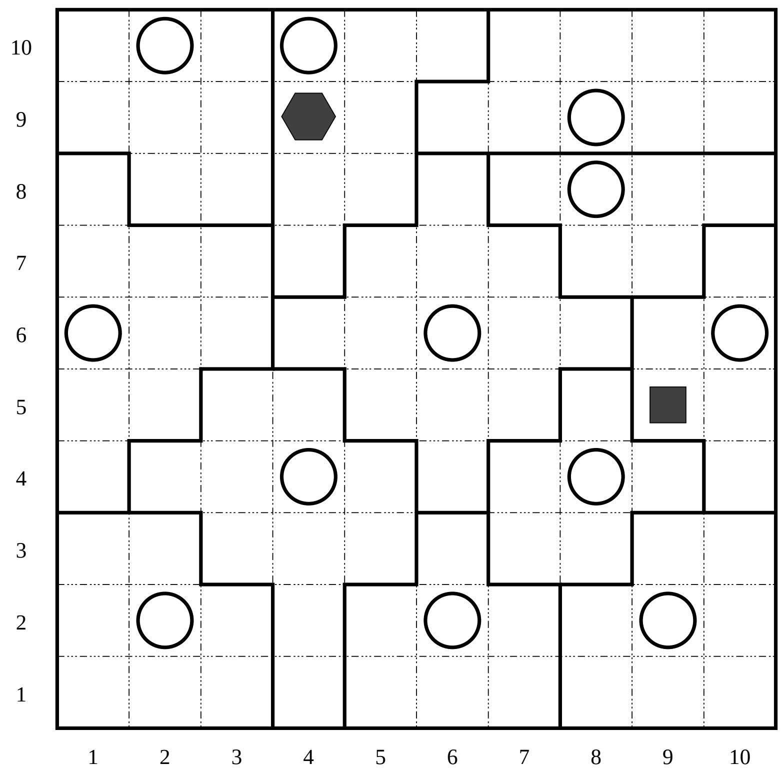


- **States:** 10,000 combinations of location of agent and location of goal.
- **Actions:** North, South, East, West. Each action succeeds with probability 0.8 and fails (moving perpendicularly) with probability 0.2.
- **Reward Function:** Cost of 1 unit for each action until goal is reached.

Value function requires representing 10,000 values (or 40,000 values using Q learning).

Example 1: Decomposition Strategy

- **Impose a set of landmark states**. Partitions the state space into Voronoi cells.
- **Constrain the policy**. The policy will go from the starting cell via a series of landmarks until it reaches the landmark in the goal cell. From there, it will go to the goal.
- **Decompose the value function**.



Example 1: Formulation of Subtasks

Subtasks:

- $\mathsf{GotoLmk}(x, l)$: Go from current location x to landmark l, where l is the landmark defining the current cell or any of its neighboring cells. $[V_1(x, l)]$
- $\mathsf{LmktoLmk}(l_1, l_2)$: Go from landmark l_1 to landmark l_2. $[V_2(l_1, l_2)]$ (uses GotoLmk as a subroutine)
- $\mathsf{GotoGoal}(x, g)$: Go from current location x to the goal location g (within current cell). $[V_3(x, g)]$

The cost of getting to the goal is now the cost of getting from x to one of the neighbor landmarks l_1, then from l_1 to the landmark in the goal cell l_g, and then from l_g to the goal g:

$$V(x, g) = min_{l \in N(NL(x))}[V_1(x, l) + V_2(l, NL(g)) + V_3(NL(g), g)]$$

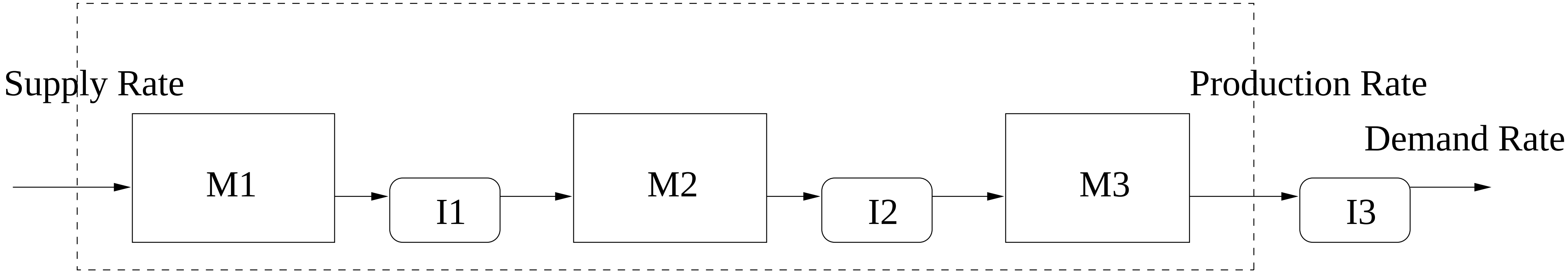
This requires only 6,070 values to store as a set of Q functions (compared to 40,000 for the original problem).

Each of these subtasks can be shared by many combinations of initial states and goal states.

Example 1: Solution Method

- **Solve all possible** GotoLmk **and** GotoGoal **subtasks.**
- **Solve** LmktoLmk **task using** GotoLmk **as a subroutine**.
- **Choose actions by using combined value function.**

Example 2: Factory Transfer Lines (Mahadevan et al.)



A **transfer line** is a pipeline of manufacturing machines. At each time t, each machine has three possible actions:

- Produce: Take inputs and produce outputs, which go into the inventory. Produce can only be performed if there are raw materials available from the previous machine.
- Maintain: Perform maintenance of the machine.
- Idle: Do nothing.

Example 2: Formulation as an MDP

- **State Variables:** current supply rate, current demand rate, inventory of each machine, amount of time each machine has operated since previous maintenance. For a 12-machine line with all values in the range 0–9, this gives 10^{25} states
- **Actions:** A vector of the actions of all the machines. For a line with 12 machines, there are 531,441 possible action vectors.
- **Reward Function:** Each finished item (at the end of the line) has a fixed positive sales value. Items in any inventory have a fixed cost per unit time. If a machine fails, there is a cost for repair. There is a (lower) cost for performing maintenance. Raw materials are free.

The real problem is more complex than this because each action takes a variable amount of time to complete.

Even this simplified version is not solvable as a flat MDP by any of our methods.

Example 2: Decomposition

Break the problem into 12 MDPs, one for each machine:

- **State Variables:** supply rate, demand rate, inventory, amount of time since machine was maintained. 10^4 states.
- **Actions:** Produce, Maintain, Idle. 3 actions.
- **Reward Function:** Each item produced gives a positive reward, each item in inventory has a fixed cost per unit time, machine repair cost, machine maintenance cost.

The only change here is that the reward function gives *each* machine a reward for producing items, rather than giving a reward for each final, finished item.

Example 2: Solution Method

- **Solve each subproblem**. This requires computing the value function for all combinations of supply and demand rates at each machine.
- **Combine the solutions online**. Every T time steps, each machine computes its supply rate and demand rate and then executes the policy corresponding to those rates for the next T time steps.

Discussion

- **Scale-up**: These decompositions allow RL methods to scale to solve much larger problems than can be solved by non-hierarchical methods.
- **Transfer**: Solutions to subproblems can be re-used/transferred between tasks.
  HDG: $\mathsf{GotoGoal}(x, g)$ can be shared by all problems where g is the goal.
  Factories: The policy for one machine can be shared by any transfer line that contains that machine (e.g., even the same transfer line with different supply and demand rates).
- **Two Main Approaches:**
  - **Single Agent Decomposition:** State space and rewards may be decomposed, but action space is not. Only one subtask is "executing" at any time.
  - **Multi-Agent Decomposition:** Action space is decomposed into a set of agents. Each agent is responsible for part of the action.

Discussion (continued)

- **Imposed Constraints**

  HDG: constrains the policy to visit the landmark states

  Factories: constrains policy to be cross-product of component policies

  These constraints lose optimality in general (unknown for factory problem), in return for finding a policy.

- **Hiding Partial Observability**. Does not arise in these examples.

Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
  - Defining Subcomponents (Tasks; Macros)
  - Learning Problems and Learning Algorithms
  - Dealing with Suboptimality
  - State Abstraction
  - Discovering Subtasks
  - Open Problems
- **Multi-Agent Decomposition**
- **Review of Key Points**

Design Issues for Single-Agent Decomposition

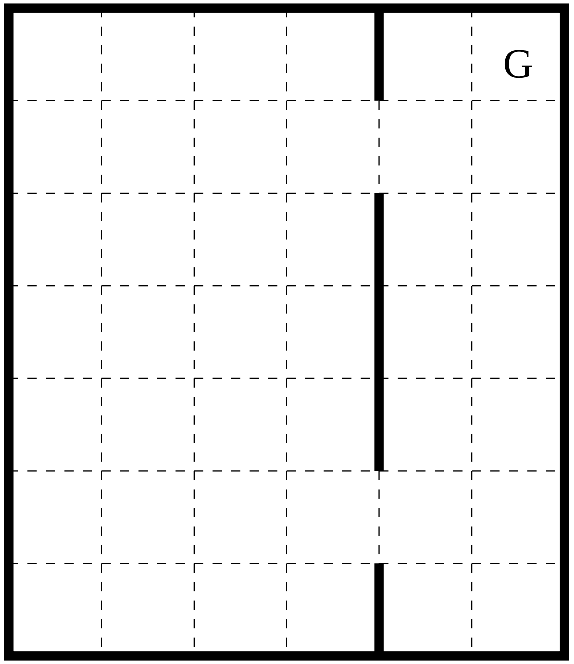
- **How to define the components?** Options, partial policies, subtasks.
- **What learning algorithms should be used?**
- **Overcoming suboptimality?** Are there any ways we can recover from the suboptimality created by our imposed hierarchy?
- **Can we employ state abstraction?** Can a subtask ignore parts of the state?
- **Automatic discovery of the hierarchy?**

Methods for Defining Subcomponents

Three basic methods have been explored for defining subcomponents:

- **Options**
- **Partial policies**
- **Subtasks**

Example problem:



- **actions:** North, South, East, West
- **rewards:** Each action costs -1. Goal gives reward of 0.

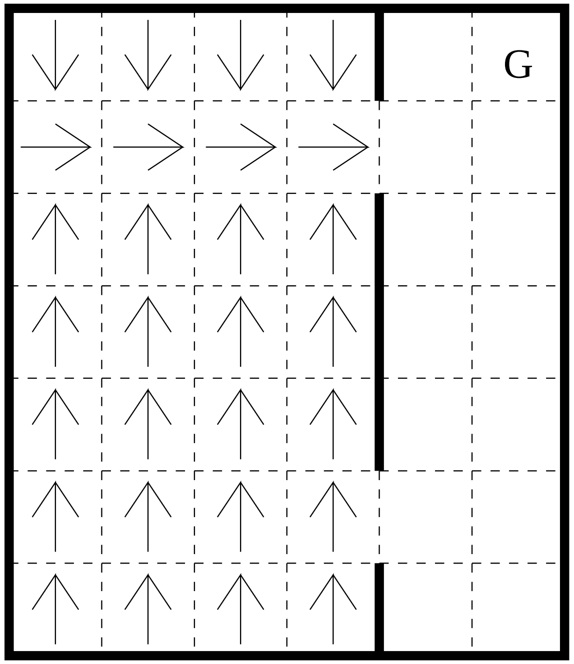
Options (Sutton; Precup; Singh)

An option is a macro action defined as follows:

- **A region of the state space**. The option can begin execution from any state in this region.
- **A policy** π. This tells for all states in the problem, what action the option will perform.
- **A termination condition**. This tells, for each state, the probability that the option will terminate if it enters that state.

Example: "Exit room by upper door"

- **Initiation region:** Any point in left room.
- **Policy:** See figure.
- **Termination condition:** Terminate with probability 1 in any state outside the room; 0 inside the room.



Partial Policies (Parr and Russell)

Partial Policy

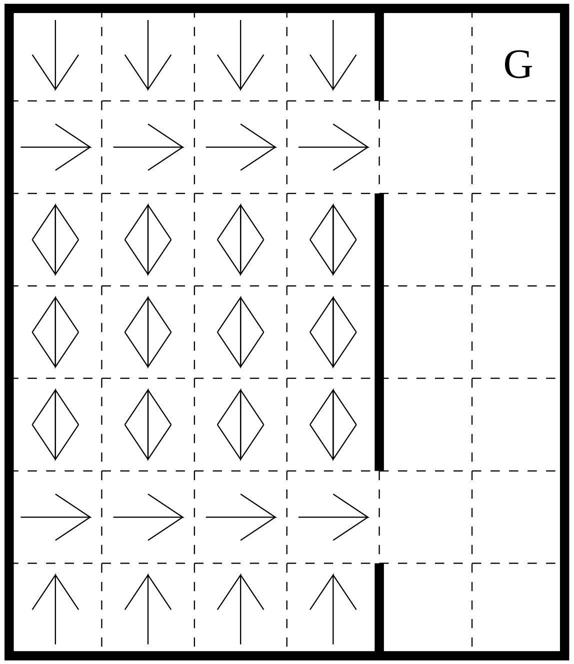
- **Mapping from states to sets of possible actions**.

  Example:

  $\pi(s_1) = \{\mathsf{South}\}$

  $\pi(s_2) = \{\mathsf{North}, \mathsf{South}\}$

Only need to learn what to do when the partial policy lists more than one possible action to perform.

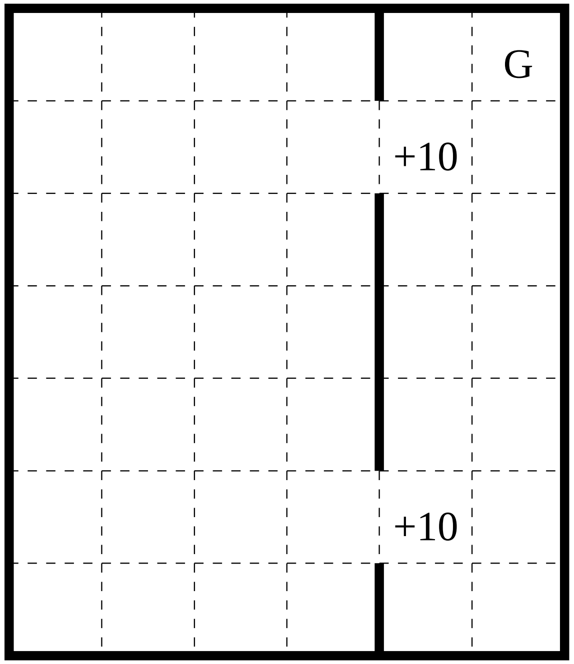


Subtasks

A subtask is defined by

- **A region of state space**. The subtask is only "active" within this region.
- **A termination condition**. This indicates when the subtask has completed execution.
- **A pseudo-reward function**. This determines the value of each of the terminal states.

Example: Exit by nearest door



- **Region:** Left room.
- **Termination condition:** Outside left room.
- **Pseudo-reward:** 0 inside left room; +10 in both "boundary states".

Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
  - Defining Subcomponents (Tasks; Macros)
  - Learning Problems and Learning Algorithms
  - Dealing with Suboptimality
  - State Abstraction
  - Discovering Subtasks
  - Open Problems
- **Multi-Agent Decomposition**
- **Review of Key Points**

Learning Problems

1. **Given a set of options, learn a policy over those options**

   Precup, Sutton, and Singh

   Kalmár, Szepesvári, and Lörincz

2. **Given a hierarchy of partial policies, learn policy for the entire problem**

   Parr and Russell

3. **Given a set of subtasks, learn policies for each subtask**

   Mahadevan and Connell

   Sutton, Precup and Singh

   Ryan and Pendrith

4. **Given a set of subtasks, learn policies for the entire problem**

   Kaelbling (HDG), Singh (Compositional Tasks), Dayan and Hinton (Feudal Q), Dietterich (MAXQ), Dean and Lin.

Task 1: Learning a policy over options

Basic idea: Treat each option as a primitive action.

Complication: The actions take different amounts of time.

Fundamental Observation: MDP + options = Semi-Markov Decision Problem (Parr)

Semi-Markov Q learning (Bradke/Duff, Parr)

- **In** s**, choose option** a **and perform it**
- **Observe resulting state** s'**, reward** r**, and number of time steps** N
- **Perform the following update:**

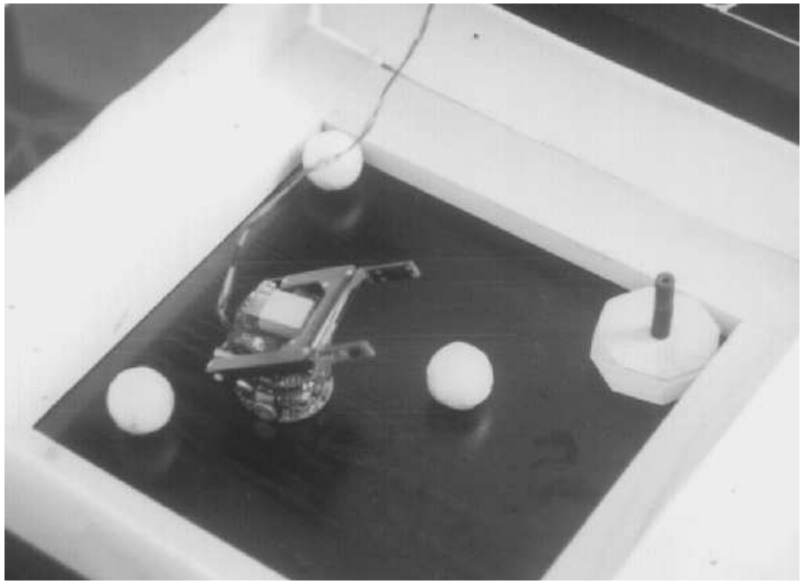
$$Q(s,a) := (1-\alpha)Q(s,a) + \alpha \cdot [r + \gamma^N \max_{a'} Q(s',a')]$$

Under suitable conditions, this will converge to the best possible policy definable over the options.

Task 1: Things to notice

- **Only need to learn Q values for a subset of the states:** All possible initial states. All states where some option could terminate.
- **Learned policy may not be optimal**.
  The optimal policy may not be representable by some combination of given options. For example, if we only have the option Exit-by-nearest-door, then this will give a suboptimal result for states one move above the level of the lower door.
- **If $\gamma = 1$ (no discounting), Semi-Markov Q learning = ordinary Q learning**
- **Model-based algorithms are possible.** The model must predict the probability distribution over the possible result states (and the expected rewards that will be received).

Task 1 Example: Kalmár, Szepesvári and Lörincz



Task: Pick up ball, carry to stick, hit against stick.

Sensors: holding-ball, linear vision (stick finder), IR proximity sensors

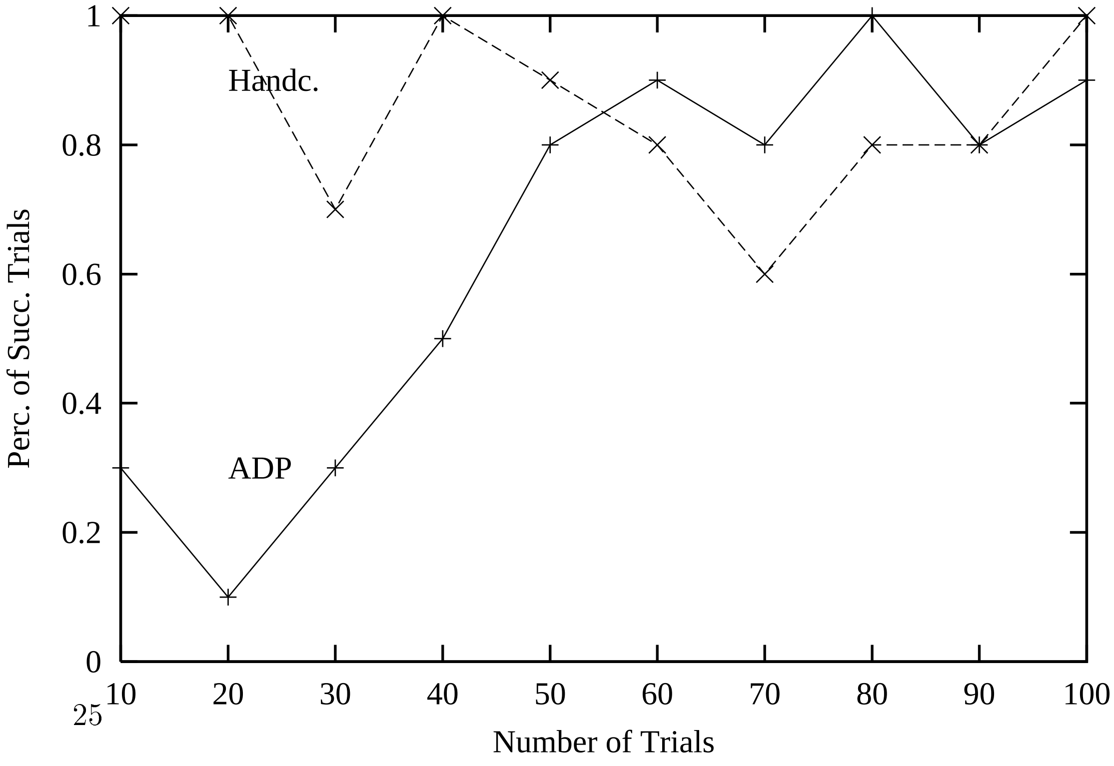
Task 1: Example (Kalmár et al. Subtasks)

Hand-coded control rules for five subtasks

- Explore avoiding obstacles
- Examine object and grasp if ball
- Goto stick
- Hit the stick
- Go backward

Individual subtasks are only partially observable. Top-level task is an SMDP.

Performance slightly exceeds best hand-coded policy:



Task 2: Learning with Partial Policies

- **Basic Idea: Execute the partial policy until it reaches a "choice state"** (i.e., a state with more than one possible action)
- **This defines a Semi-MDP**
  - **States:** all initial states and all choice state
  - **Actions:** actions given by $\pi(s)$
  - **Reward:** sum of rewards until next choice state
- **Apply Semi-Markov Q learning**
  Converges to best possible refinement of given policy.

Task 2: Hierarchical Partial Policies (Hierarchy of Abstract Machines)

Parr extended the partial policy idea to work with hierarchies of partial policies. Within a partial policy, an action can be any of:

- **Primitive action**
- **Call another partial policy as a subroutine**
- `RETURN` (return to caller)

Convert the hierarchy into a flat SMDP:

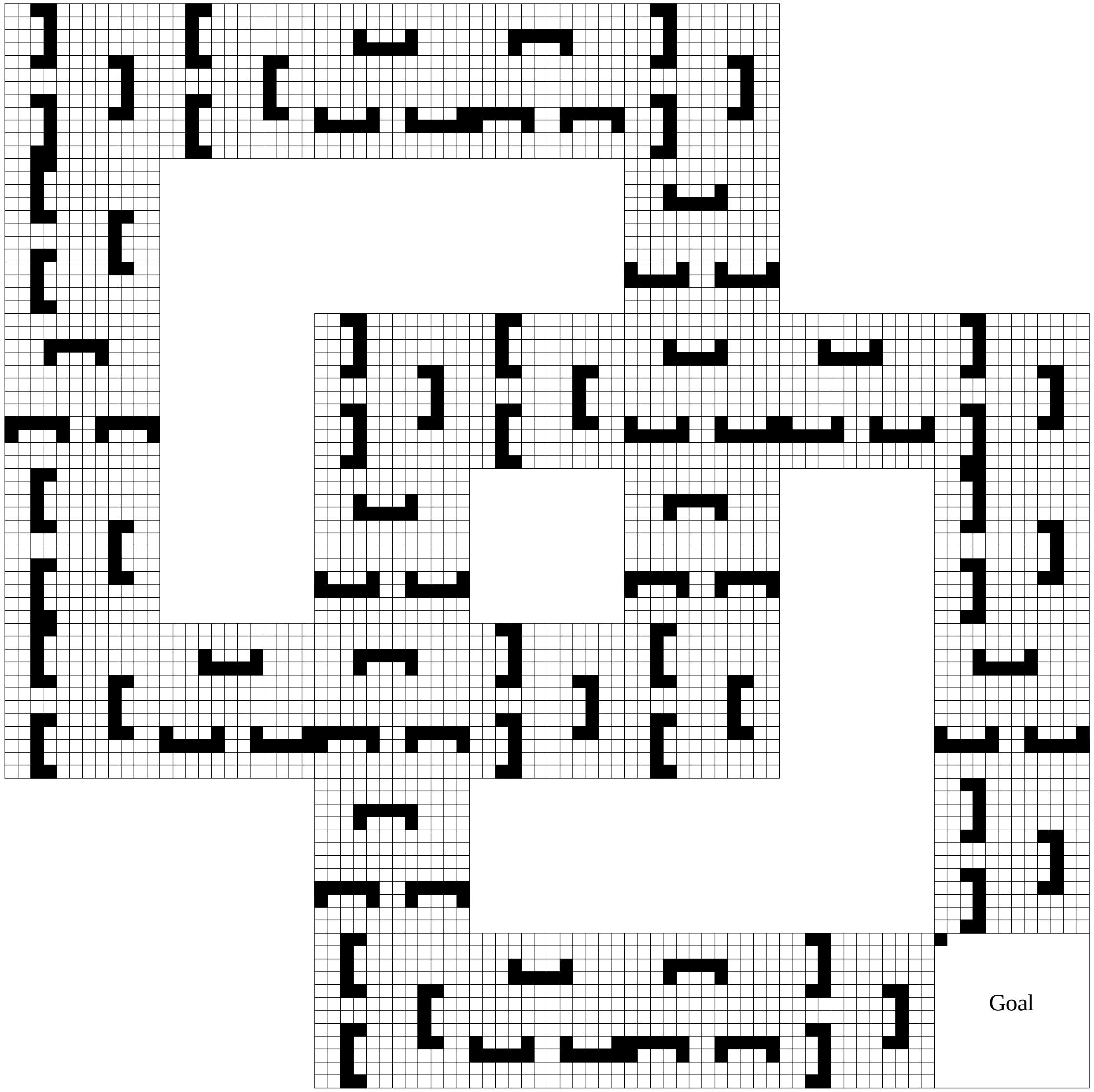
- **States:** pairs of [state, call-stack] pairs
- **Actions:** as dictated by partial policy
- **Reward function:** same as original reward function.

Apply SMDP Q learning

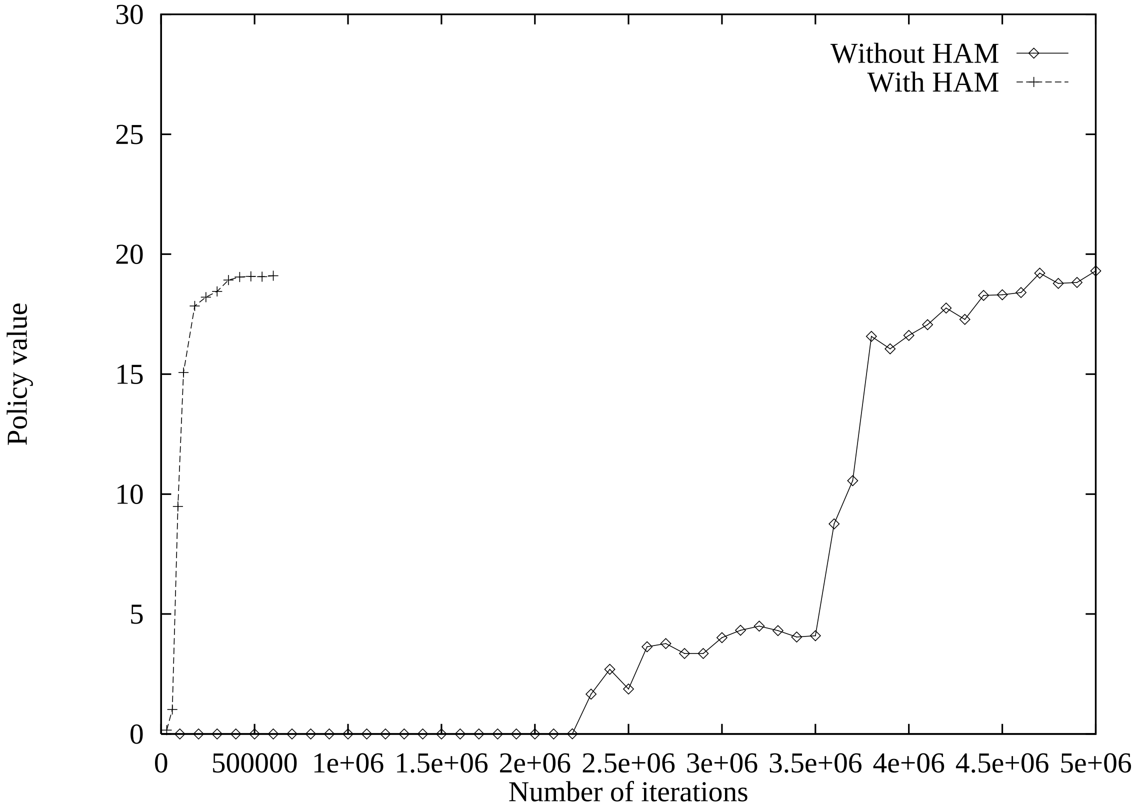
Task 2: Example: Parr's Maze Problem

Partial Policies

- TraverseHallway(d)
  calls ToWallBouncing and BackOut.
- ToWallBouncing(d_1, d_2)
  calls ToWall, FollowWall
- FollowWall(d)
- ToWall(d)
- BackOut(d_1, d_2)
  calls BackOne, PerpFive
- BackOne(d)
- PerpFive(d_1, d_2)



Task 2: Results (Parr)



Value of starting state. Flat Q learning ultimately gives a better policy.

Task 3: Learn policies for a given set of subtasks

Each subtask is an MDP

- **States:** All non-terminated states.
- **Actions:** The primitive actions in the domain.
- **Reward:** Sum of original reward function and pseudo-reward function.

Task 3: Example: Box Pushing (Mahadevan and Connell)

- **Task: Push boxes out of middle of room and against the walls**
- **Top-level Policy is Given:** Mahadevan and Connell used a simple subsumption architecture with three subtasks:
  - Find Box
  - Push Box (until it hits the wall)
  - Escape (get unstuck)

  The most important is Escape, and the least important is Find Box.
- **They defined a set of features and a local reward function for each subtask**
- **They trained the subtasks online via Q learning**. At each point in time, only one subtask is active, so that is the subtask that is learning.
- **Within each subtask, there was no problem with partial observability**

Task 3: Sharing Training Experiences (Kaelbling; Ryan/Pendrith; Sutton/Precup/Singh)

Consider two subtasks: ExitByUpperDoor, ExitByLowerDoor. Each time an action is taken in the left room, this provides training data for both tasks. Let $\tilde{R}_u$ and $\tilde{R}_\ell$ be the pseudo-reward functions for these two tasks.

Multi-task Learning:

- **In state s, choose action a and execute it**
- **Observe resulting state s' and reward r**
- **Update the Q values for each task:**

$$Q_u(s,a) := (1-\alpha)Q_u(s,a) + \alpha[r + \tilde{R}_u(s') + \gamma \max_{a'} Q_u(s',a')]$$

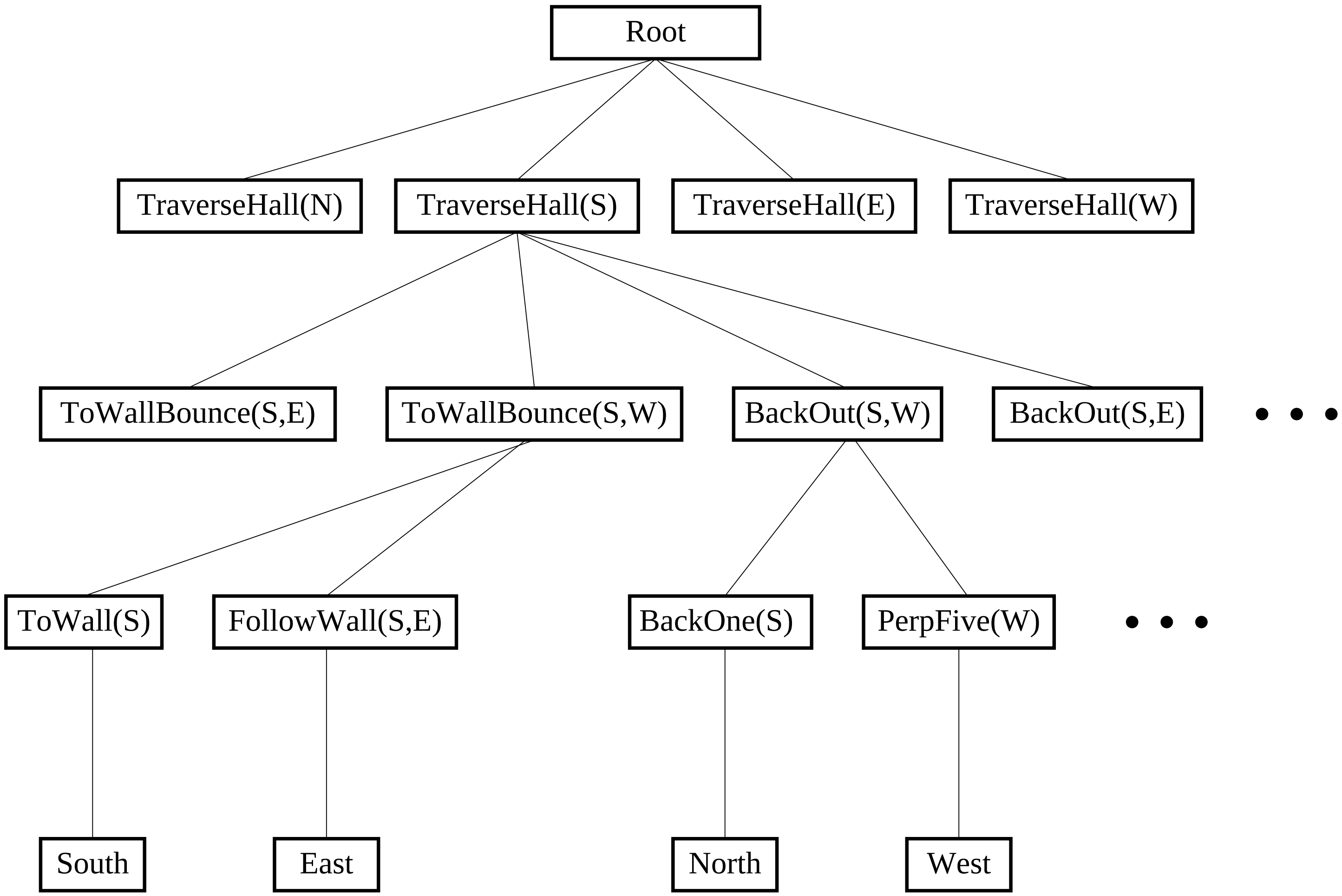
$$Q_\ell(s,a) := (1-\alpha)Q_\ell(s,a) + \alpha[r + \tilde{R}_\ell(s') + \gamma \max_{a'} Q_\ell(s',a')]$$

If exploration is sufficient, then both Q functions will converge to the optimal Q values for their subtasks.

This is only useful to the extent that the two tasks have overlapping active regions of state space.

Task 4: Learning Policies for a Hierarchy of Tasks

It is easy to define a hierarchy of tasks and subtasks.



Task 4: SMDP Q learning applied hierarchically

- **At state s inside subtask i, choose child subtask j and invoke it (recursively)**
- **When it returns, observe resulting state s', total reward r, and number of time steps N**

$$Q(i, s, j) := (1 - \alpha)Q(i, s, j) + \alpha[r + \tilde{R}_i(s') + \gamma^N \max_{a'} Q(i, s', a')]$$

If each subtask executes a GLIE policy (Greedy in the Limit with Infinite Exploration), then this will converge (Dietterich).

However, it converges to only a *locally optimal* policy.

GLIE Exploration

An *exploration policy* **is the policy executed during learning**.

An exploration policy is GLIE if

- **Each action is executed infinitely often in every state that is visited infinitely often.**
- **In the limit, the policy is greedy with respect to the** Q**-value function with probability 1.**

Examples of GLIE policies:

- **Boltzmann Exploration** with the temperature gradually decreased to zero in the limit.
- ϵ**-Greedy Exploration** with ϵ gradually decreased to zero in the limit.

GLIE exploration is needed in order to ensure that tasks higher in the hierarchy obtain training examples of the lower level tasks executing their optimal policies.

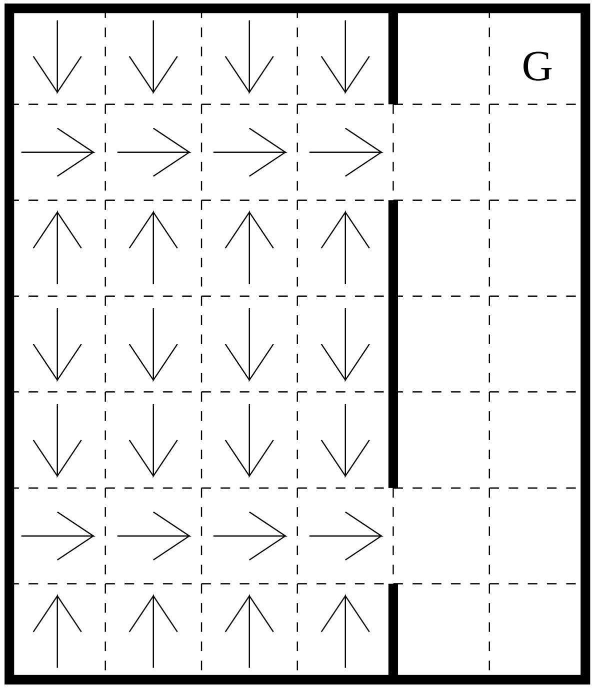
Task 4: Local Optimality Problem

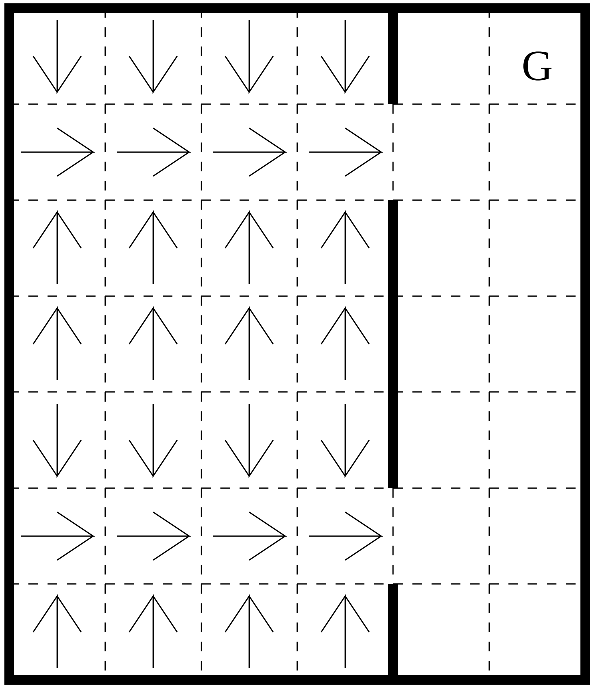
- **Any MDP may have multiple optimal policies**

  They all achieve the same expected reward.

  They may behave very differently, however.

- **Some of these may be better for the parent task**





Hierarchical Optimality versus Recursive Optimality

- **Hierarchical Optimality: The overall learned policy is the best policy consistent with the hierarchy**
- **Recursive Optimality: The policy for a task is optimal** *given* **the policies learned by its children**

- **Parr's partial policy method learns hierarchically optimal policies**. Information about the value of the result states can propagate "into" the subproblem.
- **Hierarchical SMDP Q learning converges to a recursively optimal policy**. Information about the value of result states is blocked from flowing "into" the subtask.

Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
  - Defining Subcomponents (Tasks; Macros)
  - Learning Problems and Learning Algorithms
  - Dealing with Suboptimality
  - State Abstraction
  - Discovering Subtasks
  - Open Problems
- **Multi-Agent Decomposition**
- **Review of Key Points**

Dealing With SubOptimality

Three Methods Have Been Developed:

- **Mixing Options and Primitives**
- **Re-Definition of Subtasks**
- **Non-Hierarchical Execution**

Suboptimality (1): Mixing Options and Primitives

Recall Task 1: Learn a policy to choose which option to execute in each state.

- **Include the primitive actions as "trivial options"**
- **SMDP Q learning will converge to optimal policy**
- **Issue: This increases the branching factor, and hence, may be slower than learning without options!**

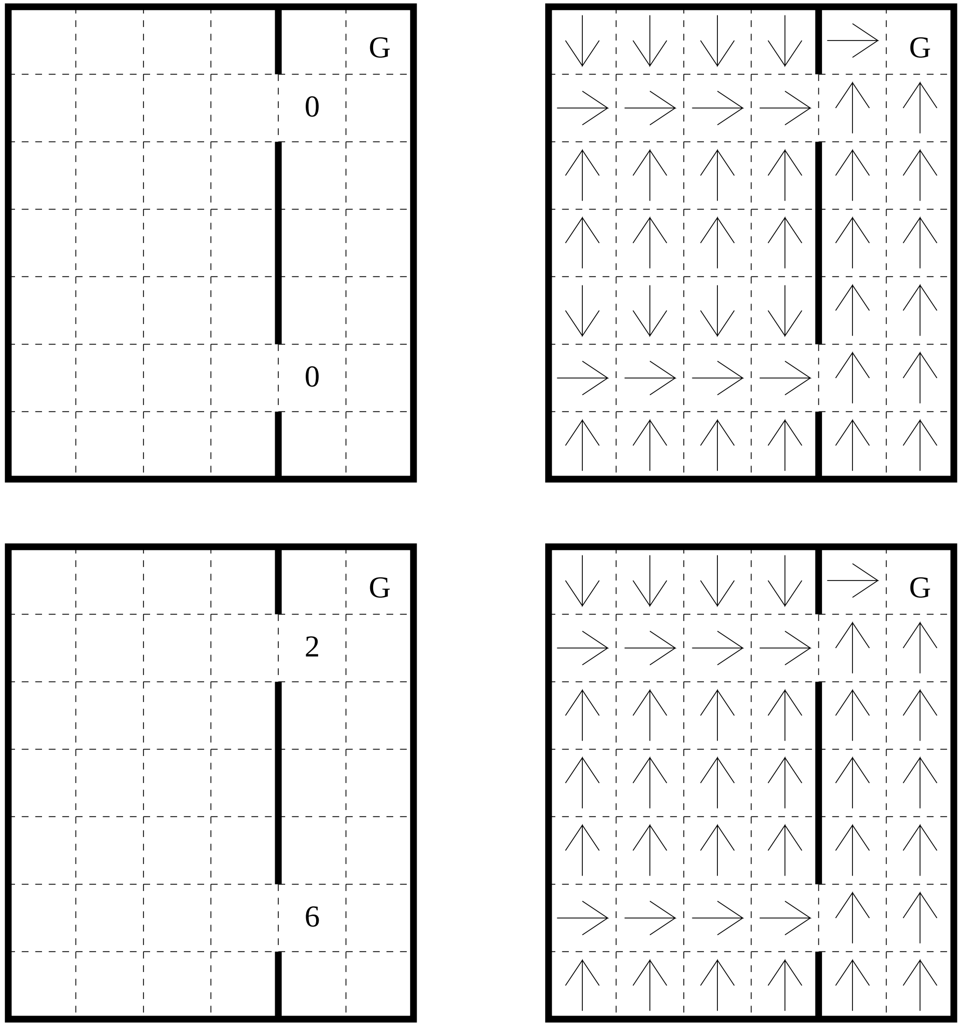
  The exact impact of options on learning depends on the properties of the learning algorithm. The options bias the exploration and backup process, but they do not constrain the set of policies.

Suboptimality (2): Dynamically Redefining the Subtasks (Dean/Lin)

At the cost of additional learning, we can overcome the local optimality problem.

- **Let** $B(i)$ **denote the "boundary" states of task** i
- **Make initial guesses** $\hat{V}(s)$ **for** $V(s)$ **for each state** $s \in B(i)$ **(for all** i**).**
- **Repeat until convergence:**
  - **Define Pseudo-reward Functions:** $\tilde{R}_i(s) = \hat{V}(s)$ for each $s \in B(i)$, zero elsewhere.
  - **Apply hierarchical SMDP Q learning** (or other appropriate method) to learn recursively optimal policy)
  - **Update the guesses for** $\hat{V}(s)$

Dynamically Redefining the Subtasks (2)



Dynamically Redefining the Subtasks (3)

Online Algorithm for Redefining Subtasks

- **At state s inside subtask i, choose child subtask j and invoke it (recursively)**
- **When it returns, observe resulting state s', total reward r, and number of time steps N**

$$\tilde{R}_j(s') := (1-\beta)\tilde{R}_j(s') + \beta \max_{a'} Q(i, s', a')$$

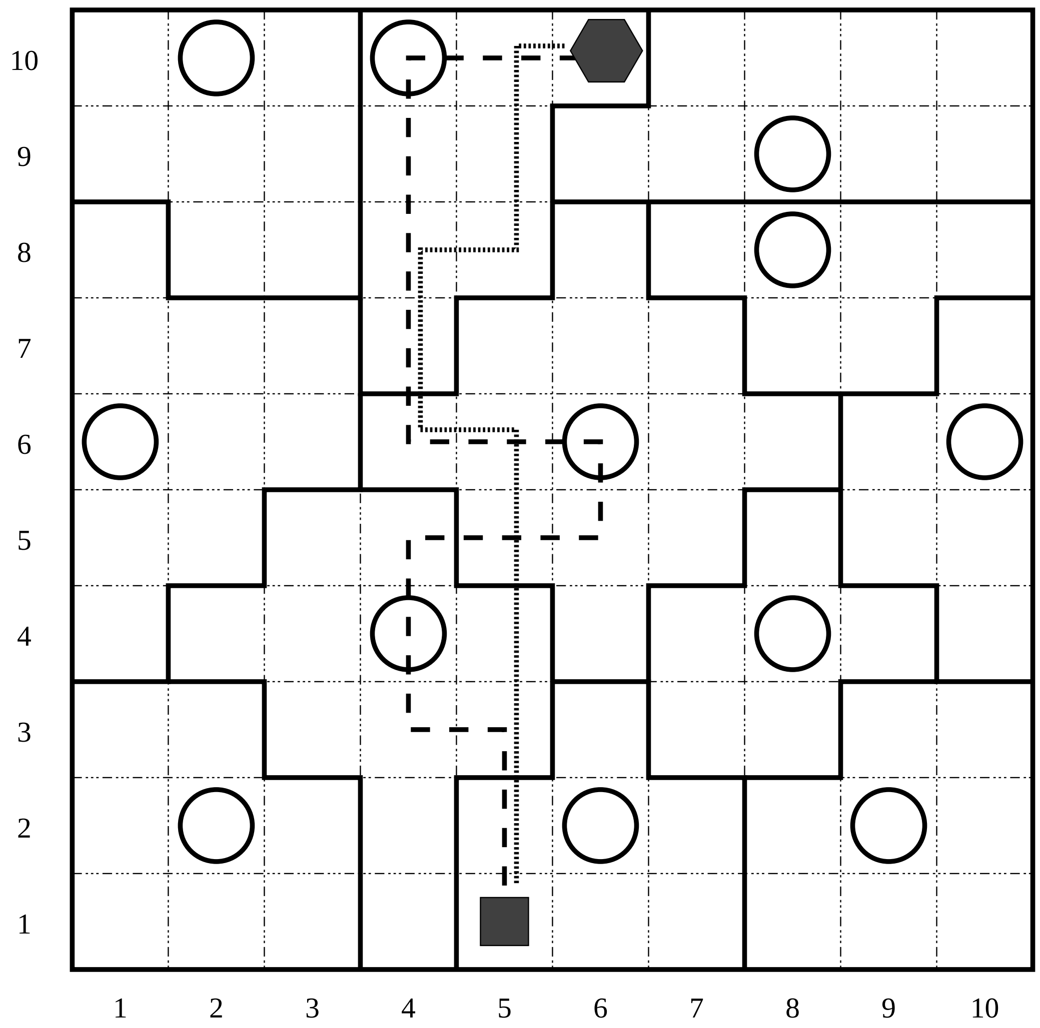
$$Q(i, s, j) := (1-\alpha)Q(i, s, j) + \alpha[r + \tilde{R}_i(s') + \gamma^N \max_{a'} Q(i, s', a')]$$

This incrementally modifies the pseudo-reward function of each subtask. This allows information to "propagate into" subtasks. This should converge to a hierarchically optimal policy (proof has not been carried out, however).

Suboptimality 3: Non-Hierarchical Execution (Kaelbling)

- **Let $Q^\pi(s,a)$ be the Q function for the learned, hierarchical policy π.** a is usually an option or a subtask
- **At each time step, compute** $a = \text{argmax}_a Q^\pi(s,a)$
  and execute *one primitive action* according to a.
- **Repeat**

This executes an option (or subtask) until another subtask is better.
"Early Termination" of options/subtasks
It is analogous to the Policy Improvement step of the Policy Iteration algorithm. It is guaranteed to improve the policy.


10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10

Suboptimality 3: Non-Hierarchical Execution (continued)

Things to Note:

- **Normally, SMDP Q learning only learns values at starting states and states where some option/subtask terminates. Hence, learning $Q(s, a)$ for all states requires extra work.**
- **For hierarchies with more than 2 levels, we need some kind of hierarchical decomposition of the value function in order to use this method most effectively.**

Suboptimality 3: The MAXQ Value Function Decomposition (Dietterich)

Key Observation: The reward function for the parent task is the value function of the child task

$$Q(i,s,j) = E\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, \pi\}$$

$$Q(i,s,j) = E\left\{\sum_{u=0}^{N-1} \gamma^u r_{t+u} \; + \; \sum_{u=N}^{\infty} \gamma^u r_{t+u} \middle| \, s_t = s, \pi \right\}$$

$$\begin{aligned} Q(i,s,j) &= \sum_{s',N} P(s',N|s,j) \cdot [V(j,s) + \gamma^N \max_{j'} Q(i,s',j')] \\ &= V(j,s) + C(i,s,j) \end{aligned}$$

Suboptimality 3: MAXQ Value Function Decomposition (2)

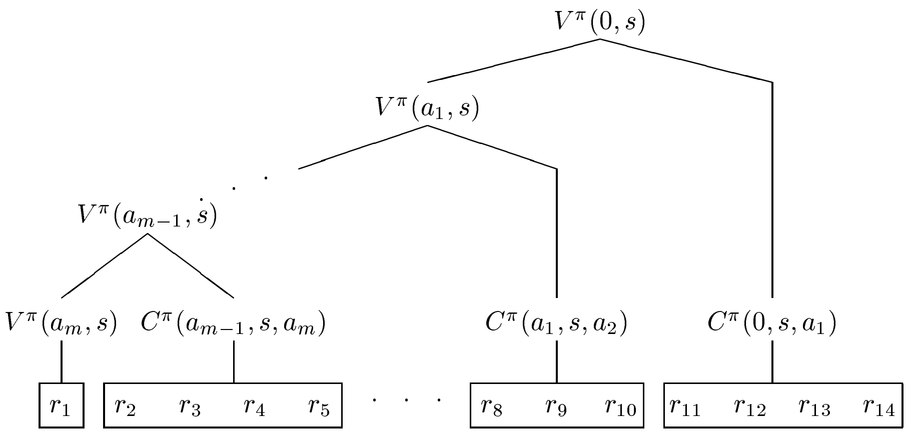
MAXQ-Q Learning: SMDP Q learning applied to the C function

- **At state s in subtask i, choose j and execute it**
- **When j returns, observe result state s' and number of time steps N and perform**

$$C(i, s, j) := (1 - \alpha_t)C(i, s, j) + \alpha_t \cdot \gamma^N [\max_{a'} V(a', s') + C(i, s', a')].$$

This converges to a recursively optimal policy if exploration is GLIE.

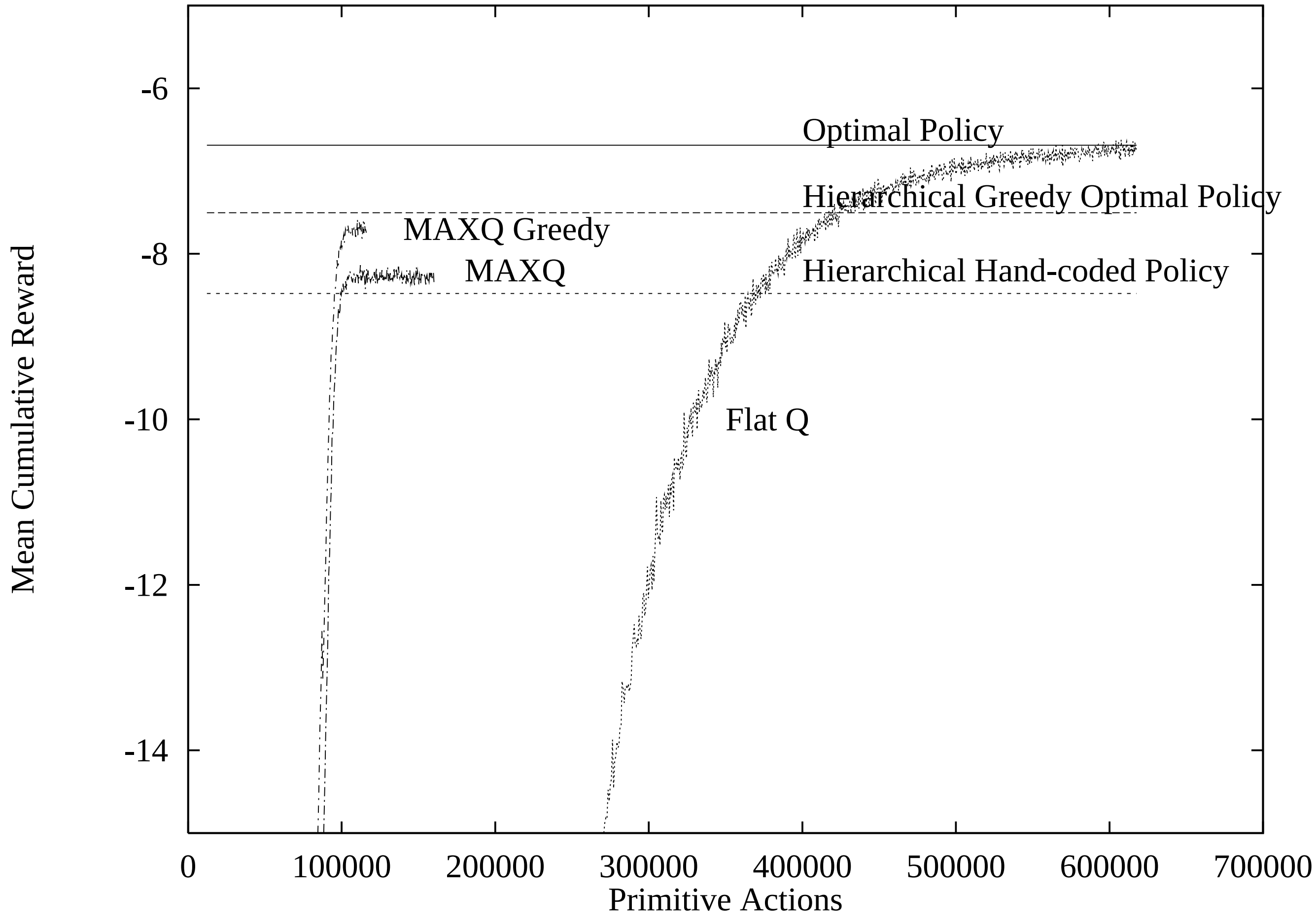
Suboptimality 3: MAXQ Value Function Decomposition (3)



$V(\mathsf{Root}, s)$ is the value of the best path from the root down to a primitive leaf action.

We obtain non-hierarchical execution by performing the primitive action which is at the tip of the best path and then recomputing the best path at each time step.

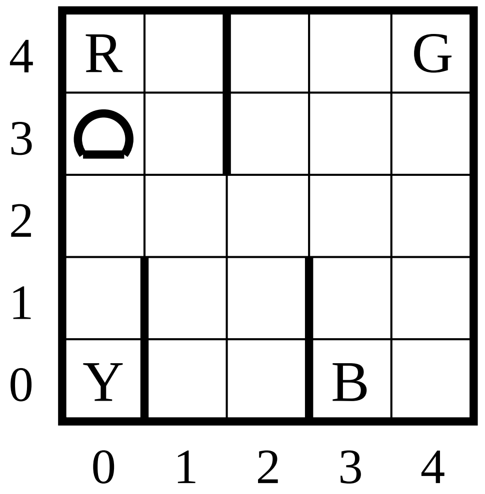
Suboptimality 3: Results for HDG

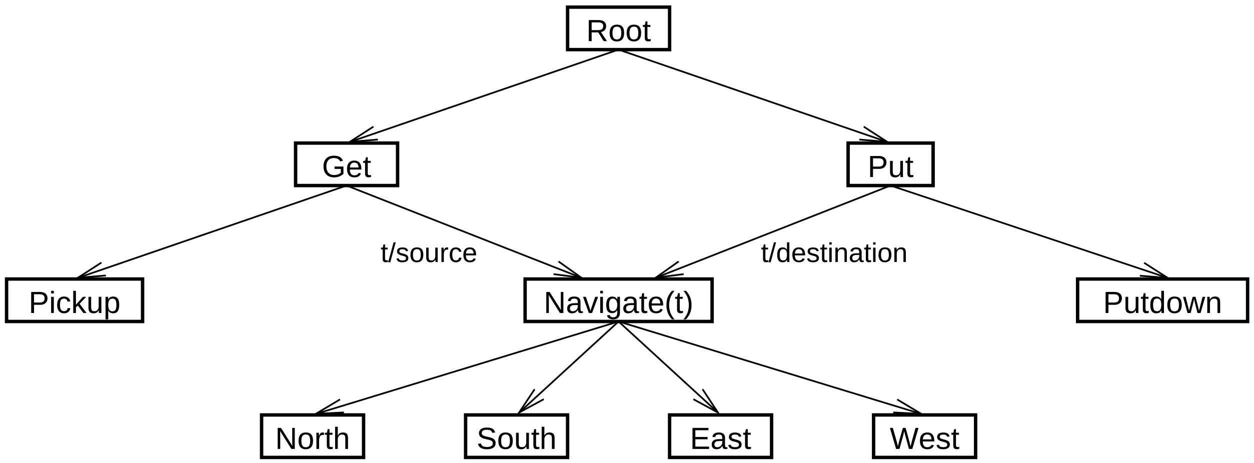


Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
  - Defining Subcomponents (Tasks; Macros)
  - Learning Problems and Learning Algorithms
  - Dealing with Suboptimality
  - State Abstraction
  - Discovering Subtasks
  - Open Problems
- **Multi-Agent Decomposition**
- **Review of Key Points**

State Abstraction: Motivating Example





- **States:** location of taxi, passenger source and destination
- **Actions:** North, South, East, West, Pickup, Putdown
- **Rewards:** -1 for each action, -10 for illegal Pickup or Putdown, $+20$ for getting passenger to destination.

Two Basic Kinds of State Abstraction

- **Independent State Variables**

  Example: Navigation depends only on taxi location and target, not on passenger source or destination

- **Funnel Actions**

  Example: Completion cost of Root task depends only on passenger source and destination, not on taxi location

Independent State Variables

Two Conditions:

- **Independence of transition probability**

  Partition the state variables into two sets X and Y such that

$$P(x', y', N|x, y, a) = P(x', N|x, a) \cdot P(y'|y, a)$$

- **Independence of reward**

$$R(x', y'|x, y, a) = R(x'|x, a)$$

If these properties hold, then Y is irrelevant for option/subtask a.

The completion function $C(i, s, a)$ can be represented using $C(i, x, a)$.

Funnel Actions (undiscounted case only)

Condition:

- **Result Distribution Irrelevance** For all s_1 and s_2 such that $s_1 = (x, y_1)$ and $s_2 = (x, y_2)$ and all s',

$$P(s'|s_1, a) = P(s'|s_2, a).$$

If this condition holds, then the state variables Y are irrelevant for option/subtask a.

$$C(i, s, a) = \sum_{s'} P(s'|s, a)[\tilde{R}(s') + \max_{a'} Q(i, s', a')]$$

The C values must be the same.

Intuition: When a is executed in s_1 or in s_2, it takes the environment to the same set of possible result states s' with the same probabilities. Hence, the completion cost is the same.

State Abstraction with MAXQ (Dietterich)

- **Independent State Variables**

  This allows us to ignore some state variables within a subtask, because our local C function can still be represented.

- **Funnel Actions**

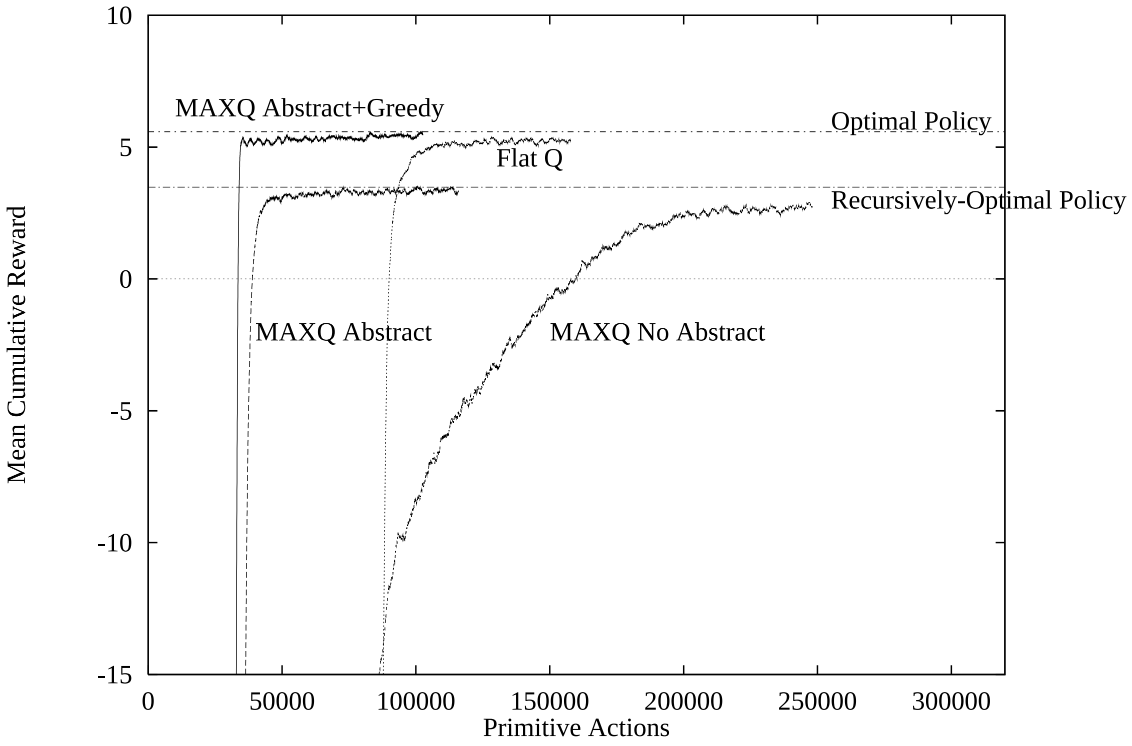
  This allows us to ignore state variables in a parent task, because regardless of the details of the child task, the environment will end up in the same set of states.

Configuration	Number of Values
Flat Q Learning	3000
MAXQ (no abstraction)	14,000
MAXQ (state abstraction)	632

MAXQ Q Learning Converges Under State Abstraction

MAXQ-Q learning converges to a recursively optimal policy assuming that state abstractions are applied under the two conditions defined above. (Dietterich)

State Abstraction is very important for achieving reasonable performance with MAXQ-Q learning



Tradeoff Between Abstraction and Hierarchical Optimality

Claim: To obtain state abstraction, there must be a barrier between the subtask and the context. Information must not cross that barrier.

- Suppose that the optimal policy within a subtask depends on the relative value of two states s_1 and s_2 outside of that subtask.
- Then information about the difference between s_1 and s_2 must flow into the subtask, and the representation of the value function within the subtask must depend on the differences between s_1 and s_2.
- This could defeat any state abstraction that satisfies the two conditions given above.
- Therefore, this dependency must be excluded from the subtask (e.g., by a pseudo-reward function)
- But to achieve hierarchical optimality, we must have this kind of information flow. Therefore, state abstraction and hierarchical optimality are incompatible.

Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
  - Defining Subcomponents (Tasks; Macros)
  - Learning Problems and Learning Algorithms
  - Dealing with Suboptimality
  - State Abstraction
  - $\boxed{\text{Discovering Subtasks}}$
  - Open Problems
- **Multi-Agent Decomposition**
- **Review of Key Points**

Discovering Subtasks

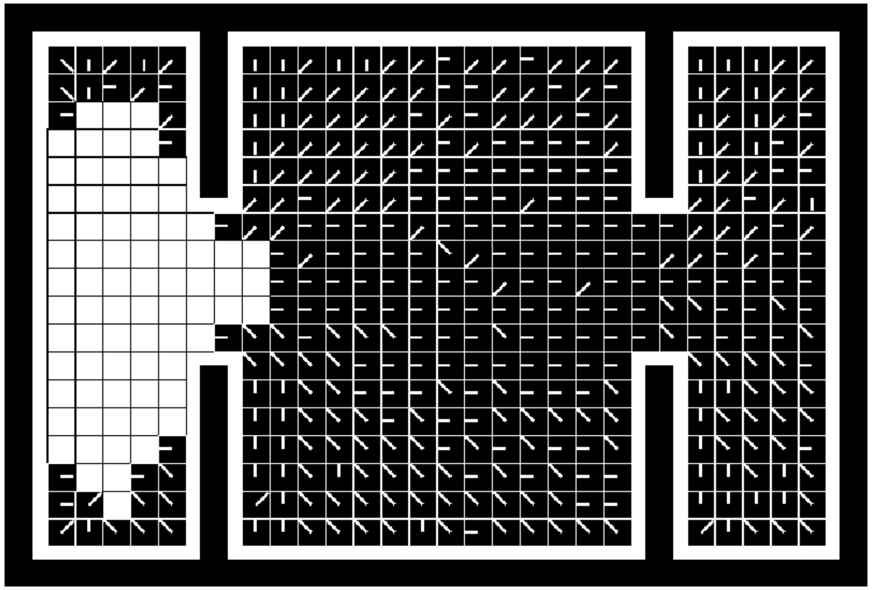
Basic Idea:

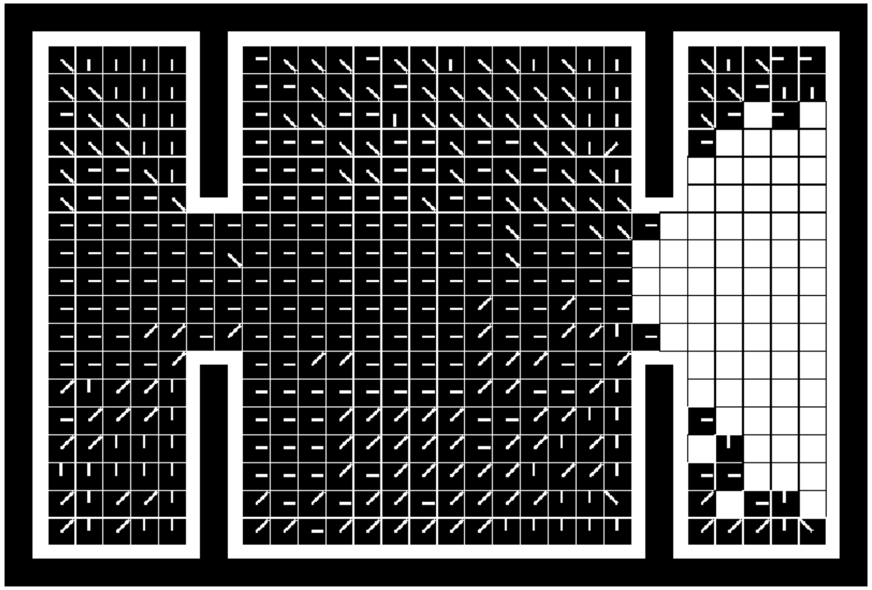
- **Solve a set of closely related tasks.**
- **Identify shared subcomponents.**

Example: Thrun and Schwartz

- **Simultaneously solve a set of related tasks**
- **Four data structures are learned:**
  - $Q(t, s, a)$ for each task t
  - $\pi(k, s)$ the policy for "skill" k
  - $m(k, s)$ the "domain" of the skill $\{0, 1\}$
  - $u(t, k)$ probability that task t uses skill k
- **Objective function encourages the tasks to use skills** (even sacrificing some optimality to do this).

Discovering Subtasks: Results from Thrun and Schwartz





Start and goal states are randomly positioned in the rooms at opposite ends of the world. Skills are learned for traversing the hallway in each direction.

Skills are executed probabilistically: At each state s, with probability $\propto u(t, k)$, the action $\pi(k, s)$ is executed. Otherwise the action $\mathrm{argmax}_a\, Q(t, s, a)$ is executed.

Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
  - Defining Subcomponents (Tasks; Macros)
  - Learning Problems and Learning Algorithms
  - Dealing with Suboptimality
  - State Abstraction
  - Discovering Subtasks
  - $\boxed{\text{Open Problems}}$
- **Multi-Agent Decomposition**
- **Review of Key Points**

Open Problems in Single-Agent Decomposition

- **Model-Based Algorithms**. Model-free methods are slow. Sutton/Precup/Singh have an online algorithm for learning models of options that is a good start in this direction.
- **Non-Asymptotic Theoretical Results**. All results thus far are asymptotic.
- **Bounds on Loss of Optimality**. Can we bound the loss in optimality resulting from a given decomposition? From recursive optimality versus hierarchical optimality?
- **Function Approximation**. All results assume tabular representations of the Q or C function.
- **Discovering and Modifying Subtasks/Options**. We lack a good formal model of what we are trying to accomplish (but see Kalmar/Szepesvari for one proposal).

Outline

- **Introduction: Purpose of Hierarchical RL**
- **Single-Agent Decomposition**
- $\boxed{\textbf{Multi-Agent Decomposition}}$
  - Defining Subtasks
  - Solution Methods
  - Open Problems and Speculations
- **Review of Key Points**

Multi-Agent Decomposition

- **Mahadevan et al.:** Factory Transfer Line: Each machine is a separate agent.
- **Crites/Barto:** Elevator scheduling: Each elevator is a separate agent.
- **Schneider et al.:** Power distribution: Each node in the network is a separate agent.
- **Meuleau et al:** Bombing Campaigns: Each target is a separate agent.
- **Boyan/Littman:** Q-Routing. Each router in a packet-switched network is a separate agent.

Defining Subtasks

- **Key Idea: Factor the action space**
  The action space consists of a vector of the actions chosen by individual agents.
- **Factor State Space**. Each agent sees only relevant local state (except for Crites' elevators, which see the global state)
- **Factor Reward Function**. Each agent receives local rewards/penalties. Mahadevan defines a pseudo-reward function. (Crites' elevators see global reward function.)

The same issue arises of the relationship between local optimality and global optimality.

Solution Methods

- **Independent solution followed by merging**.
  Mahadevan, Meuleau. Coupling between subtasks is resolved at merge time in a straightforward way.
- **Simultaneous Online Solution**
  - **With no communication** (Crites)
  - **Communicating the value function** (Schneider)
    Each node in the network takes a local average of its neighbors' value functions in the current state and propagates this backward.
  - **Communication of Q values** (Boyan/Littman)
    Each router acknowledges the receipt of a packet by sending back its Q value for that packet.
- **Independent Solution Followed by Solving the Original Problem**.
  Singh/Cohn describe a method for using the subproblem solutions to guide the solution of the original problem.

Open Problems

- **Theory**. We have essentially no analytical framework and no results.
- **General-purpose algorithms with provable performance.**

Curiously, we do have more real-world applications than for the single-agent case!

Speculation: Economics may be a Source of Ideas

Should agents interact through artificial economies? (Baum)

- Mahadevan's machines could buy their inputs and sell their outputs. This would provide a justification for the pseudo-reward function.
- Schneider's substations could buy their input power and sell their output power. This would be more sensible than the current "local averaging" scheme.
- Boyan/Littman's packets could rent their communication links.

Can results from economics be applied to bound the loss in global optimality under local optimality?

Review of Key Points

- **Central Problem:** How to decompose a complex MDP into subcomponents such that (a) the components are easy to solve and (b) their solutions can be combined to give near-optimal performance on the original MDP.
- **Single-Agent Decomposition:**
  - Well understood.
  - SMDP Q learning works in all cases (with slight modifications).
  - To define a subtask, you must "guess" the values of the result states.
  - There is a tradeoff between hierarchical optimality and the use of state abstraction.
  - There are several methods for recovering from the suboptimality created by the hierarcuhy.
- **Multi-Agent Decomposition:**
  - Poorly understood—but more practical successes.
  - Open issue: What is communicated among agents? How does this lead to good results?

Bibliography

References

Baum, E. B. (1998). Manifesto for an evolutionary economics of intelligence. Tech. rep., NEC Research Institute, Princeton, NJ.

Boyan, J. A., & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In Cowan, J. D., Tesauro, G., & Alspector, J. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 6, pp. 671–678. Morgan Kaufmann, San Francisco.

Bradtke, S. J., & Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov decision problems.. In Tesauro et al. (Tesauro, Touretzky, & Leen, 1995), pp. 393–400.

Crites, R. H., & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In Touretzky, D. S., Mozer, M. C., & Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 8, pp. 1017–1023. The MIT Press, Cambridge.

Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems, 5*, pp. 271–278. Morgan Kaufmann, San Francisco, CA.

Dean, T., & Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. Tech. rep. CS-95-10, Department of Computer Science, Brown University, Providence, Rhode Island.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *Fifteenth International Conference on Machine Learning*, pp. 118–126. Morgan Kaufmann.

Dietterich, T. G. (1999). Hierarchical reinforcement learning with the MAXQ value function decomposition. Tech. rep., Oregon State University.

Hauskrecht, M., Meuleau, N., Boutilier, C., Kaelbling, L., & Dean, T. (1998a). Hierarchical solution of Markov decision processes using macro-actions. Tech. rep., Brown University, Department of Computer Science, Providence, RI.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998b). Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pp. 220–229 San Francisco, CA. Morgan Kaufmann Publishers.

Kaelbling, L. P. (1993). Hierarchical reinforcement learning: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 167–173 San Francisco, CA. Morgan Kaufmann.

Kalmár, Z., Szepesvári, C., & Lörincz, A. (1998). Module based reinforcement learning for a real robot. *Machine Learning*, *31*, 55–85.

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, *55*(2-3), 311–365.

Meuleau, N., Hauskrecht, M., & Kim, K.-E. (1998). Solving very large weakly coupled markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 165–172 Cambridge, MA. MIT Press.

Parr, R. (1998). *Hierarchical control and learning for Markov decision processes*. Ph.D. thesis, University of California, Berkeley, California.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, Vol. 10 Cambridge, MA. MIT Press.

Precup, D., & Sutton, R. (1998). Multi-time models for temporally abstract planning. In *Advances in Neural Information Processing Systems*, Vol. 11 Cambridge, MA. MIT Press.

Schneider, J., Wong, W.-K., Moore, A., & Reidmiller, M. (1999). Distributed value functions. In *Proceedings of the Sixteenth International Conference on Machine Learning* San Francisco, CA. Morgan Kaufmann.

Singh, S., & Cohn, D. (1998). How to dynamically merge Markov decision processes. In *Advances in Neural Information Processing Systems*, Vol. 11 Cambridge, MA. MIT Press.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, *8*, 323.

Sutton, R., & Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA.

Sutton, R. S., Precup, D., & Singh, S. (1998). Between MDPs and Semi-MDPs: Learning, planning, and representing knowledge at multiple temporal scales. Tech. rep., University of Massachusetts, Department of Computer and Information Sciences, Amherst, MA.

Tesauro, G., Touretzky, D. S., & Leen, T. K. (Eds.). (1995). *Advances in Neural Information Processing Systems*, Vol. 7. The MIT Press, Cambridge.

Thrun, S., & Schwartz, A. (1995). Finding structure in reinforcement learning.. In Tesauro et al. (Tesauro et al., 1995), pp. 385–392.

Wang, G., & Mahadevan, S. (1999). Hierarchical optimization of policy-coupled semi-Markov decision processes. In *Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann.