

 NUS | Computing

Classical Planning

CS4246/CS5446
AI Planning and Decision Making

This lecture will be recorded!

1

Solving Planning Problems

- **Planning Problem or Model**
 - Appropriate abstraction of states, actions, effects, and goals
- **Planning Algorithm**
 - Input: a problem
 - Output: a solution in the form of an action sequence (a plan)
- **Planning Solution**
 - A **plan** or **path** from the initial state(s) to the goal state(s)
 - Any path
 - A **goal state** that satisfies certain properties

2

Classical Planning

- **Definition:**
 - Find a sequence of actions to achieve a goal in an environment that is:
 - Discrete
 - Deterministic
 - Static
 - Fully observable
- **Main challenges:**
 - How to represent the planning problem? – PDDL from STRIPS
 - How to search for a solution (plan)? – Search and satisfiability
 - How to use heuristics to solve for a solution (plan) more efficiently?

3

Planning Model

PDDL and STRIPS Planning

4

Planning Domain Definition Language

- **PDDL**
 - Derived from the original STRIPS planning language
 - Logic-based, **lifted** propositional representation (to restricted FOL)
 - Basic version for classical planning, extensions in active research
- **Planning problem representation**

	Examples
State	$At(P_1, SFO) \wedge At(P_2, SIN)$
Action	$Action(Fly(P_1, SFO, SIN))$ PRECOND: $At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(SIN)$ EFFECT: $\neg At(P_1, SFO) \wedge At(P_1, SIN)$
Goal	$At(p, SIN) \wedge Plane(p)$

5

Main Characteristics

- **Factored representation – Why?**
 - Represent state of the world as a set of **state variables** or **fluents**
 - Each fluent depicts an **aspect** of the world that changes with time
- **Domain independent heuristics**
 - Avoid need for domain specific heuristics to work well in planning
- **On-demand state computation**
 - “Implicit” transition function
 - Just specify the current state
 - Define actions that can compute state-transitions
 - Use actions to generate the other states as needed

6

5

6

State Representation

- State**
 - A conjunction of **ground atomic fluents**
 - Ground atomic – there is a single predicate, with only constant arguments; function-free
 - Examples:
 - Hungry \wedge Sleepy*
 - New(Plane₁) \wedge Safe(Plane₁)*
 - At(Plane₁, SIN) \wedge At(Plane₂, SFO)*
- Database semantics:**
 - Closed-world** assumption – Any fluents that are not mentioned are False
 - e.g., *Fierce(CS4246, Lecturer)*
 - Unique names** assumption – Any two objects with different names are different
 - e.g., *Plane₁* and *Plane₂*

7

7

Goal Representation

- Goal State:**
 - Goal g is a partially specified state, represented as a conjunction of literals
 - A state s **satisfies** a goal g if s contains all the literals in g
 - Examples:
 - Hungry \wedge Sleepy \wedge Bored* satisfies the goal *Hungry \wedge Bored*
 - At(Cargo₁, SFO)* satisfies the goal *At(c , SFO)* with substitution $\{c/Cargo_1\}$ where c is a variable for any cargo

Follow general definition of Goal in PDDL on Slide 14

8

8

Action Schema

- Defines a family of ground actions**
 - All variables assumed to be universally quantified
 - Lifts propositional logic to restricted subset of first-order logic
- Pre-Conditions:**
 - Conjunctions of literals (positive and negative atomic sentences) defining the applicable states in which the action can be executed
- Effects:**
 - Conjunctions of literals defining the result of executing the action

Action(Fly(p , from, to))
 PRECOND: $At(p, \text{from}) \wedge Plane(p) \wedge Airport(\text{from}) \wedge Airport(\text{to})$
 EFFECT: $\neg At(p, \text{from}) \wedge At(p, \text{to})$

9

9

Grounded Actions

Action(Fly(P_1 , SFO, SIN))
 PRECOND: $At(P_1, \text{SFO}) \wedge Plane(P_1) \wedge Airport(\text{SFO}) \wedge Airport(\text{SIN})$
 EFFECT: $\neg At(P_1, \text{SFO}) \wedge At(P_1, \text{SIN})$

Action(Fly(P_2 , SIN, SFO))
 PRECOND: $At(P_2, \text{SIN}) \wedge Plane(P_2) \wedge Airport(\text{SIN}) \wedge Airport(\text{SFO})$
 EFFECT: $\neg At(P_2, \text{SIN}) \wedge At(P_2, \text{SFO})$

10

10

Action Execution

- Current state and available action:
 $At(P_1, \text{SFO}) \wedge Plane(P_1) \wedge Airport(\text{SFO}) \wedge Airport(\text{SIN})$
Action(Fly(P_1 , SFO, SIN))
 PRECOND: $At(P_1, \text{SFO}) \wedge Plane(P_1) \wedge Airport(\text{SFO}) \wedge Airport(\text{SIN})$
 EFFECT: $\neg At(P_1, \text{SFO}) \wedge At(P_1, \text{SIN})$
- Execute action
 $Fly(P_1, \text{SFO}, \text{SIN})$
- Remove from database of states
 $At(P_1, \text{SFO})$
- Add to database of states
 $At(P_1, \text{SIN})$
- Updated state:
 $At(P_1, \text{SIN}) \wedge Plane(P_1) \wedge Airport(\text{SFO}) \wedge Airport(\text{SIN})$

11

11

Action Effects

- Applicable State**
 - A ground action a is **applicable** in state s if s **entails** the precondition of a
- Execution Result**
 - Result of executing applicable action a in state s is a state s' formed by:
 - Removing fluents in action effects that are negative literals - **delete list** $DEL(a)$, and
 - Adding fluents in action effects that are positive literals - **add list** $ADD(a)$
$$s' = (s - DEL(a)) \cup ADD(a)$$
 - New states are generated **on demand** with the $ADD(a)$ and $DEL(a)$ functions as a result of execution an action

12

12

The Frame Problem

- Definition**
 - What changes and what stays as the **result of the action**
 - Needs to be addressed by any real-world planning models
- Action representation in PDDL:**
 - Times and states are **implicit** in action schemas:
 - Preconditions always refers to time t and the effect to time $t + 1$
 - Specifies result of an action in terms of changes; everything that stays the same unmentioned
- State representation in PDDL:**
 - Uses **closed world assumption** that anything not mentioned is FALSE
 - The fluents do not explicitly refer to time

13

13

Planning Problem and Solution

- Initial state**
 - A state – a conjunction of ground atoms
 - Example: $A \wedge t(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(SIN) \wedge \dots$
- Goal**
 - A conjunction of literals that may contain variables
 - Any variables are treated as **existentially quantified**
 - Example: $At(p, SIN) \wedge Plane(p)$.
- STRIPS planning restricts Goal to positive literals with no variables**
- Problem solution**
 - A sequence of actions that end in a state s that entails the goal
 - Example: Any plan that reaches state: $Plane(P_1) \wedge At(P_1, SIN)$ entails goal $At(p, SIN) \wedge Plane(p)$

14

14

Example: Air Cargo Transport Planning

Init($At(C_1, SFO) \wedge At(C_2, SIN) \wedge At(P_1, SFO) \wedge At(P_2, SIN)$
 $\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
 $\wedge Airport(SIN) \wedge Airport(SFO)$)
Goal($At(C_1, SIN) \wedge At(C_2, SFO)$)

Action($Load(c, p, a)$,
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p)$)
Action($Unload(c, p, a)$,
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p)$)
Action($Fly(p, from, to)$,
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to)$)

Planning Domain
Planning Problem

Adapted from source: RN Figure 11.1
15



15

Planning as State-Space Search

Forward (progression) search
Backward (regression) search

16

16

Planning as State-Space Search

- Map planning problem into search problem in state-space**
 - States are ground states with only binary fluents
 - Goal state has all the positive fluents in problem's goal
 - Applicable actions in a state are ground actions
 - Different planning procedures have different search spaces
- Planning search tree :**
 - Each **node** is a state, including initial state and goal state
 - Each **branch** is an action that allows a transition from one state to another
 - A plan is a **path** from the initial state to the goal state in search tree
- Algorithms:**
 - Forward (progression) state-space search
 - Backward (regression) relevant-states search

17

17

Planning by Forward Search

Start at a state s
Repeat:

- Stop if goal g is satisfied**
 - Check if the goal is a subset of the state s 's description: $g \subseteq s$
- Compute applicable actions**
 - Check if pre-conditions of an action a is a subset of the state s 's description: $Precond(a) \subseteq s$
- Compute successor states**
 - Add the list of positive effects of an action a to the state s 's description; delete the list of negative effects: $s' = (s - DEL(a)) \cup ADD(a)$
- Pick a successor state s' as current state**

18

18

Planning by Backward Search

Start at goal g

Repeat:

1. **Stop if initial state s is satisfied**
 - Check if the initial state s is a subset of the goal state g description: $s \subseteq g$
2. **Compute relevant actions**
 - Check if effects of an action a is a subset of the goal state g description: $\text{Effects}(a) \subseteq g$
3. **Compute predecessor states**
 - Add the preconditions and remove the list of positive and negative effects of an action a to the current goal state g description

$$\text{POS}(g') = (\text{POS}(g) - \text{ADD}(a)) \cup \text{POS}(\text{Precond}(a))$$

$$\text{NEG}(g') = (\text{NEG}(g) - \text{DEL}(a)) \cup \text{NEG}(\text{Precond}(a))$$
4. **Pick a predecessor state g' to be current goal state**

19

State Descriptions in Regression

Notes

- A set of **relevant** states to consider at each step (cf. forward search)
- Regression from a **state description** to a predecessor state description

State vs Description

- In a state, every variable is assigned a value.
- For n ground fluents, there are 2^n ground states
- For n ground fluents, there are 3^n descriptions
 - Each fluent can be **positive**, **negative**, or **not mentioned**.
 - E.g. for goal: $\neg \text{Hungry} \wedge \text{Sleepy}$ describes states where **Hungry** is False and **Sleepy** is True, but other unmentioned fluents can have any value.
- A **description** represent a set of states.

20

Relevant Actions in Regression

- **Relevant actions**
 - Those that can be the last step in a plan leading up to current goal state
 - At least one of its effects must **unify** with an element of the goal;
 - Must not have any effect that negates the elements of the goal
 - Substitute **most general unifier** to keep branching factor down but not rule out any solution
- **Regression**
 - Given a goal g and an action a , regression from g over a gives a state description g' whose positive and negative literals are given by:

$$\text{Pos}(g') = (\text{Pos}(g) - \text{Add}(a)) \cup \text{Pos}(\text{Precond}(a))$$

$$\text{Neg}(g') = (\text{Neg}(g) - \text{Del}(a)) \cup \text{Neg}(\text{Precond}(a))$$

21

Relevant Actions in Regression

More formally:

- Assume a goal description g that contains a goal literal g_i and an action schema A
- If A has an effect literal e'_j where $\text{Unify}(g_i, e'_j) = \theta$ and where we define $A' = \text{Subst}(\theta, A)$ and if there is no effect A' that is the negation of a literal in g , then A' is a relevant action towards g

22

Planning as Boolean satisfiability

SATPlan

23

Planning as Boolean Satisfiability

- **Transform classical planning problem into a Boolean satisfiability (SAT) problem**
 - Translate planning problem into propositional formula in Conjunctive Normal Form (CNF)
 - Then determine whether propositional formula is **satisfiable**
 - Example:

$$(P \vee Q) \wedge (\neg Q \vee R \vee S) \wedge (\neg R \vee \neg P)$$
 Does there exist a model i.e., an assignment of truth values to the proposition that makes the formula true?
- **SAT is a NP-complete problem**
 - SAT solvers are effective in practice; can solve problems with millions of variables and constraints
 - Use domain independent heuristics to search for a solution (See RN 7.7.4)

24

The SATPlan Algorithm

```

function SATPLAN(init, transition, goal,  $T_{\max}$ ) returns solution or failure
  inputs: init, transition, goal, constitute a description of the problem
     $T_{\max}$ , an upper limit for plan length

  for  $t = 0$  to  $T_{\max}$  do
    cnf  $\leftarrow$  TRANSLATE-TO-SAT(init, transition, goal,  $t$ )
    model  $\leftarrow$  SAT-SOLVER(cnf)
    if model is not null then
      return EXTRACT-SOLUTION(model)
    return failure
  
```

Assignment of values to variables

Source: RN Fig 7.22

25

Planning Problem Translation

1. **Propositionalize the actions**
 - For each action schema, form ground propositions by substituting constants for each of the variables
2. **Add action exclusion axioms**
 - Assert that no two actions can occur at the same time
3. **Add precondition axioms**
 - For each ground action A^t , add the axiom $A^t \Rightarrow PRE(A)^t$, i.e., if an action is taken at time t , then the preconditions must have been true

26

25

26

Planning Problem Translation

4. **Define the initial state**
 - Assert F^0 for every fluent F in the problem initial state
 - Assert $\neg F^0$ for every fluent not mentioned in the initial state
5. **Propositionalize the goal**
 - Goal becomes a disjunction over all its ground instances; variables are replaced by constants
6. **Add successor state axioms**
 - For each fluent F , add an axiom of the form $F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$, where *ActionCausesF^t* stands for a disjunction of all the ground actions that add F and *ActionCausesNotF^t* stands for a disjunction of all the ground actions that delete F

27

The Frame Problem

- **The Frame Problem:**
 - Most actions leave most fluents unchanged.
 - For each action, assert an axiom for each fluent the action leaves unchanged.
- **Successor-state axioms:**
 - A clever encoding that handles the frame problem well.
 - For m actions and n fluents, need $O(mn)$ such axioms
 - In contrast, with successor-state axioms, only need $O(n)$ axioms: each axiom is longer, but only involves actions that have an effect on the fluent
- **SATPlan**
 - Find possible reachable models specifying future action sequences
 - Guarantee to find shortest path if one exists
 - Works only for fully observable or sensorless environments
 - Can find models with impossible action; memory requirement is still combinatorially large

28

28

Summary

- **Classical planning**
 - Goal oriented action planning (STRIPS Planning)
 - State, action, goal representations in PDDL
- **State-space search**
 - Forward or progressive search
 - Backward or regressive search
- **Planning as satisfiability**
 - Planning as propositional satisfiability problem
 - The SATPlan algorithm
- **Other classical planning approaches**
 - **Planning graph** – specialized data structure to encode constraints and derive heuristics
 - **Situation calculus** – describing planning problem in first-order logic
 - **Constraint satisfaction problems (CSP)** – encode bounded planning problem
 - **Partial-order planning** – represents plan as a graph; handles independent subproblems, explainability

29

Characteristics of Planning

30

29

30

Planning Algorithm Properties

- **A planning algorithm or planner is sound**
 - For any plan generated by the planning algorithm, this plan is guaranteed to be a solution
- **A planning algorithm or planner is complete**
 - If a solution exists then the planning algorithm will return a solution
- **Question: How “good” are the solutions?**
 - For any plan generated by the planning algorithm, can the plan be guaranteed to be an **optimal** solution?
 - Yes if admissible heuristics are used in the search for the solution

31

Complexity of Planning

- **Two decision problems:**
 - PlanSAT –whether there exists any plan that solves a planning problem
 - Bounded PlanSAT –whether there is a solution of length k or less
 - For classical planning, both problems are decidable as number of states is finite
- **In general:**
 - Both problems are in PSPACE; solvable with polynomial amount of space
 - In many domains, Bounded PlanSAT is NP-complete; PlanSAT is in P
 - Optimal planning is usually hard, but sub-optimal planning is sometimes easy
- **In real world planning:**
 - Agents usually asked to find plans in specific domains, not in worst-case instances
 - To do well, need good **heuristics** or **alternate planning models**

32



Real World Planning and Acting

CS4246/CS5446
AI Planning and Decision Making

1

Heuristics for Planning

Improving Efficiency

2

Heuristics for Planning

- Heuristic function
 - $h(s)$ estimates distance from a state s to the goal g
- Main idea
 - Find **admissible heuristic** for distance
 - A* or other heuristic search can then find **optimal** solutions
- Approach
 - Define a **relaxed problem** that is easier to solve
 - Exact cost of solution to easier problem is **heuristic** for original problem

3

Types of Heuristics

- Definition and application
 - Facilitated by factored representation for states and action schemas
- Search problem
 - A graph with states as nodes and actions as edges
 - Find a path connecting initial state to goal state
- Domain independent heuristics –Why?
 - Two ways to **relax** the problem (make it strictly easier):
 - Add more **edges** – **easier to find path**
 - Group multiple **nodes** together – **abstract with fewer states and easier to search**
 - Prune away irrelevant branches of search tree

Real-world problems use a combination of domain-independent and domain-specific heuristics

4

Heuristics of Adding Edges

- Ignore preconditions heuristic
 - Drop all preconditions from actions
- Ignore selected preconditions heuristic
 - Derive simpler measures of distance from goal
- Ignore delete list heuristic
 - Allow monotonic progress toward goal
- Caveats:
 - Finding solution to relaxed problem is still NP-hard
 - Trade-off in optimality or admissibility sometimes required
 - Expensive to calculate heuristics
 - Do not reduce state-space size

5

Example: 8-Puzzle as Planning

- Planning as path search

Initial	Goal																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>2</td><td>3</td><td></td></tr> <tr><td>1</td><td>8</td><td>4</td></tr> <tr><td>7</td><td>6</td><td>5</td></tr> </table>	2	3		1	8	4	7	6	5	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>1</td><td>2</td><td>3</td></tr> <tr><td>8</td><td>4</td><td></td></tr> <tr><td>7</td><td>6</td><td>5</td></tr> </table>	1	2	3	8	4		7	6	5
2	3																		
1	8	4																	
7	6	5																	
1	2	3																	
8	4																		
7	6	5																	

• Game objective:

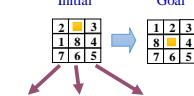
- To rearrange a given initial configuration (state) of eight numbered tiles arranged on a 3×3 board into given final or goal configuration (state)

Action($Slide(t, s_1, s_2)$)
 Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
 Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

6

Example: 8-Puzzle as Planning

- States**
 - A state description specifies the location of each of the eight tiles in one of the nine squares
- Actions or operators**
 - Tile **slides** left, right, up, or down
- Goal test**
 - State matches the goal configuration
- Path cost**
 - Each step cost 1, so path cost is just the length of the path

Initial: 

Goal: 

Action(Slide(t, s_1, s_2)
Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$)

7

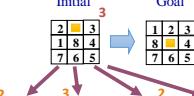
Ignore Preconditions Heuristic

- Main idea:**
 - All actions become applicable in all states
 - Any single goal fluent can be achieved in one step (if there is an applicable action)
 - No. of steps required to solve relaxed problem \approx no. of unsatisfied goals
 - Some actions may achieve multiple goals
 - Some actions may undo the effects of others
 - Possible accurate heuristic – consider 1 and ignore 2

8

Ignore Selected Preconditions Heuristic

- Remove Preconditions**
 - $Blank(s_2) \wedge Adjacent(s_1, s_2)$
- What does the heuristic do?**
 - No. of misplaced tiles heuristic
- What are the estimates?**

Initial: 

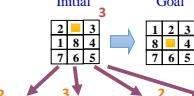
Goal: 

Action(Slide(t, s_1, s_2)
Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$)

9

Ignore Selected Preconditions Heuristic

- Remove Preconditions**
 - $Blank(s_2)$
- What does the heuristic do?**
 - Manhattan-distance heuristic
- What are the estimates?**
- Caveat:**
 - Unclear which preconditions can be selectively ignored in general

Initial: 

Goal: 

Action(Slide(t, s_1, s_2)
Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$)

10

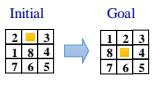
Ignore Delete List Heuristic

- Main idea:**
 - Assume only positive literals in all plans and actions
 - Remove delete list from all actions (all negative literals from effects) to make monotonic progression toward goal
 - Create a relaxed version of original problem that is easier to solve, where solution length will serve as good heuristics
 - Approximate solution can be found in polynomial time by hill-climbing

11

Ignore Delete Lists Heuristic

- Ignore delete lists**
 - How does the problem become easier when ignore delete lists heuristic is applied to this schema?
 - Goals are never undone
 - Blanks always increase

Initial: 

Goal: 

Action(Slide(t, s_1, s_2)
Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
Effect: $On(t, s_2) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$)

12

Domain-independent Pruning

- Symmetry reduction
 - Prune all symmetric branches of the search tree except for one
 - For many domains, efficiently solve intractable problems
- Forward pruning
 - Accept risk of pruning away an optimal solution, in order to focus search on promising branches
- Rule-out negative interactions
 - A problem has **serializable subgoals** if there exists an order of subgoals such that the planner can achieve them in that order without having to undo any of the previously achieved subgoals

13

13

Serializable Subgoals

- Planning examples in the real world:
 - Build a tower of blocks on Table, in any order
 - Switch on all n lights with independent switches, in any order
 - Remote Agent Planner that commands NASA's Deep Space One spacecraft (1998) – serializable by design
 - able to command spacecraft in real-time

14

14

Heuristics of State Abstraction

- State abstraction:
 - Many-to-one mapping from states in the ground representation of the problem to the abstract representation
- Main approach:
 - Ignore some fluents
 - Solution in abstract state space will be shorter than a solution in the original space (assumes **admissible heuristic**)
 - Abstract solution extensible to solution for original problem

15

15

Decomposition

- Main idea:
 - Divide problem into parts, solving each part independently, and then combining the parts - Key idea in defining heuristics
- How to choose the right abstraction to reduce total cost?
 - Defining abstraction, doing abstract search, mapping abstraction back to original problem
 - Can the cost be less than original planning cost?
 - Example:
 - Pattern databases – cost of creation amortized over many problem instances

16

16

Planning Cost with Abstraction

- Problem definition:
 - Suppose the goal is a set of fluents G , divided into disjoint subsets G_1, \dots, G_n .
 - Find plans P_1, \dots, P_n that solve the respective subgoals
 - What is the estimated cost of the plan for achieving all of G ?
- Heuristic estimation:
 - Think of each $\text{Cost}(P_i)$ as a heuristic estimate
 - If each subproblem uses an admissible heuristic, taking Max is admissible
- Assuming subgoal independence:
 - Sum the cost of solving each independent subgoal; if admissible, Sum is better than Max – Why?
 - Solution **optimistic**: when there are negative interactions between subplans for each subgoal; e.g., action in one subplan deletes goals in another subproblem.
 - Solution **pessimistic** (not admissible) when there are positive interactions; e.g., actions in one subplan achieves goals in another subproblem

17

17

Example: FF- FastForward

- Characteristics:
 - Forward state-space searcher making use of effective heuristics
 - Ignore-delete-list heuristic with graph plan for heuristic estimation
 - Hill-climbing search (modified to keep track of plan) with heuristic to find a solution
 - Non-standard hill-climbing algorithm: avoids local maxima by running a breadth-first search from the current state until a better one is found
 - If this fails, FF switches to greedy best-first search instead
 - [Hoffmann, 2001]

18

18

Hierarchical Planning and Acting

Manage complexity

19

Managing Complexity in Planning

- Hierarchical decomposition**
 - Division of **tasks** into different **subtasks** at next level
 - At each level focus only a small number of tasks
- Deferred planning**
 - Planning can occur before and during plan execution
 - Particular action can remain at an abstract level prior to the execution phase

20

Hierarchical Decomposition

- Key benefits**
 - At each level of hierarchy, a **task** is reduced to a small number of **subtasks** or **activities** at the next lower level
 - Computational cost of finding the correct way to arrange activities for current problem is small
- Examples**
 - Software components, subroutines
 - Military, government, and corporations

21

Hierarchical Task Networks

- Hierarchical task networks (HTNs)**
 - A formalism to help understand hierarchical decomposition
 - A planning model that manages complexity through task abstractions
- Key concept**
 - High-level actions (HLAs)**
- Assumptions**
 - Full observability
 - Deterministic
 - Availability of primitive actions with standard precondition-effect schemas
 - Main ideas are general in problem solving and planning and decision making

22

HTN Planning

- Planning Problem or Model**
 - HLAs, action schemas, initial state, goal state, task list
- Planning Algorithm**
 - Input: a problem
 - Output: a solution in the form of an action sequence
- Planning Solution**
 - Any executable plan generated by recursively applying:
 - HLA to nonprimitive tasks
 - Actions to primitive tasks
 - A goal state that satisfies certain properties

23

High-Level Actions (HLAs)

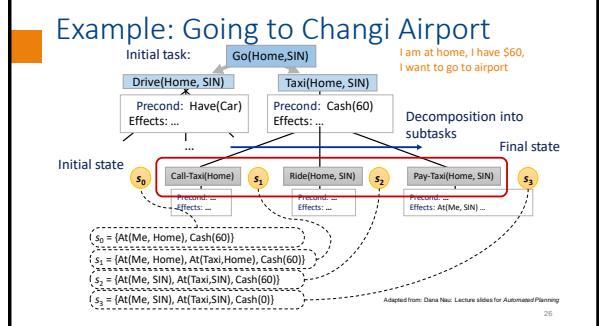
- Definition**
 - Each HLA has one or more **refinements** into a sequence of actions
 - Each (refined) action can be an HLA or a primitive action
 - Recursive refinement may be needed
- Meaning**
 - HLAs and their refinements embody knowledge about **how to do things**
e.g., Go(Home, SIN) – drive or take a taxi

24

Implementation

- HLA implementation**
 - An HLA refinement that contains only primitive actions
- High-level plan implementation**
 - High-level plan – a sequence of HLAs
 - Concatenate implementations of each HLA in the sequence
- Observation**
 - Given the precondition-effect definitions of each primitive action, can directly determine whether any given implementation of a high-level plan achieves the goal.

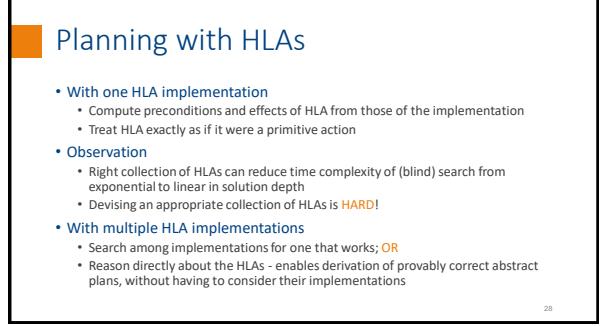
25



Planning with HLAs

- Definition**
 - Achieves the goal from a given state if at least one of its implementations achieves the goal from that state
- Note**
 - Not all implementations need to achieve the goal
 - The agent **decides** which implementation to execute
- Question:**
 - How is this different from **nondeterministic** planning?

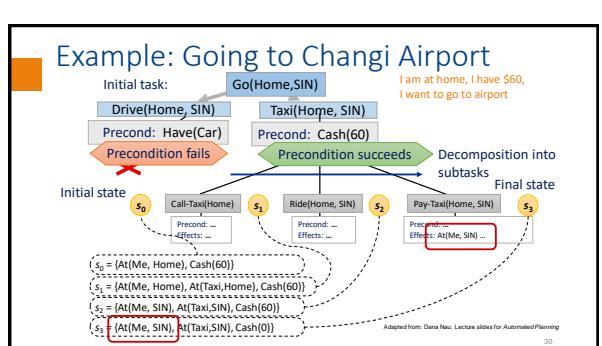
27



Hierarchical Planning as Search

Searching for Primitive Solutions

29



Searching for Primitive Solutions

- HTN Planning
 - Start with top level action Act
 - Find an implementation of Act that achieves the goal
- Hierarchical planning algorithm
 - Repeatedly choose an HLA in current plan and replace with refinement
 - Until the plan achieves the goal
- Example:
 - Breadth-first search tree
 - Plans are considered in order of depth of nesting of the refinements, rather than number of primitive steps
 - Can use graph-search, depth-first, and iterative deepening

31

Generic Planning Framework

- Classical planning definition:
 - For each primitive action a_i :
 - Provide one refinement of Act with steps – $[ai, Act]$
 - Create recursive definition of Act to add actions
 - Final refinement:
 - steps – empty, precondition – goal, effect – null
- Algorithm:
 - Repeatedly choose an HLA in the current plan
 - Replace it with one of its refinements
 - Until the plan achieves the goal

32

Hierarchical Search

```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution or failure
  frontier ← a FIFO queue with [Act] as the only element
  while true do
    if IS-EMPTY(frontier) then return failure
    plan ← POP(frontier) // chooses the shallowest plan in frontier
    hla ← the first HLA in plan, or null if none
    prefix,suffix ← the action subsequences before and after hla in plan
    outcome ← RESULT(problem,INITIAL_,prefix)
    if hla is null then // so plan is primitive and outcome is its result
      if problem.IS-GOAL(outcome) then return plan
    else for each sequence in REFINEMENTS(hla,outcome,hierarchy) do
      add APPEND(prefix,sequence,suffix) to frontier
```

Source: RN Figure 11.8

33

Hierarchical Search

- Main idea:
 - Explore space of sequences that conform to knowledge in the HLA library about how things are to be done
 - Knowledge encoded in action sequences in each refinement and in the preconditions of the refinements
- Practical impact:
 - Can generate huge plans with little search
 - e.g., O-PLAN to develop production plans for HITACHI (Bell and Tate 1995)
 - Hierarchically structured – easier for human to understand
 - [Find out more about recent applications in use!](#)

34

Proving Plan Properties

Searching for Abstract Solutions

35

Searching for Abstract Solutions

- Approach
 - Write precondition-effect description of the HLAs
 - Prove that the high-level plan achieves the goal
 - Work in small search space of high-level actions
 - Refine committed plan to achieve exponential reduction

36

Searching for Abstract Solutions

- Downward refinement property (of HLA descriptions)
 - Through description of the steps:
 - Every high-level plan that “claims” to achieve the goal achieves the goal
 - At least one implementation achieves the goal
- Main challenges
 - How to write HLAs with downward refinement property?
 - How to write HLAs with multiple implementations?
 - How to describe effects of an action that can be implemented in many different ways?
- Key idea
 - Determine if **reachable sets** of a sequence of HLAs in the plan overlap with goals

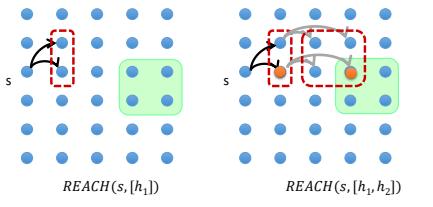
37

Reachable Set

- Reachable set of an HLA
 - Given a state s and an HLA h : $REACH(s, h)$ is the set of states **reachable by** any of the HLA's implementations
- Reachable set of a sequence of HLAs
 - Reachable set of a sequence of HLAs $[h_1, h_2]$ is the union of all the reachable sets obtained by applying h_2 in each state in the reachable set of h_1 :
$$REACH(s, [h_1, h_2]) = \bigcup_{s' \in REACH(s, h_1)} REACH(s', h_2)$$

38

Example



39

High-Level Planning

- Practical implications
 - Agent can choose element of the reachable set it ends up in when it executes the HLA
 - HLA with multiple refinements is more “powerful” than the same with fewer refinements
- High-level plan
 - A sequence of HLAs
 - Achieves goal if its reachable set intersects set of goal states
 - Otherwise, the plan does not work
- Search algorithm
 - Search among high-level plans
 - Look for one whose reachable set intersects goal
 - Once that happens, commit to that abstract plan
 - Focus on refining the plan further

40

Representing HLA Effects

- Effects as reachable sets
 - As reachable set for each possible initial state
 - Represent changes made to each fluent or state variable
- Recall: Primitive action
 - Can add or delete a fluent or variable or leave it unchanged
- HLA
 - Can also control variable value, depending on implementation chosen
 - Description derivable, in principle, from descriptions of its refinements, such that the downward refinement property holds

41

Representing Reachable Set

- Notations:
 - \sim means possibly, if the agent chooses
 - E.g., \tilde{A} means “possibly add A ”, i.e., either leave A unchanged or make it True
- Questions:
 - What do \tilde{A} and $\tilde{\tilde{A}}$ mean?
- Example
 - $Go(Home, SIN)$ with two refinements
 - $Drive(Home, SIN)$ and $Taxi(Home, SIN)$
 - Possibly delete $Cash$ (if agent decides to take a taxi)
 - So should have effect \tilde{Cash}

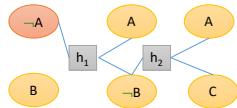
42

41

42

Example

- Consider:
 - Schemas for HLAs h_1 and h_2 :
 - Action (h_1 , Precond: $\neg A$, Effect: $A \wedge \neg B$)
 - Action (h_2 , Precond: $\neg B$, Effect: $\neg A \wedge \neg C$)
- Meaning:
 - h_1 adds A and possibly delete B
 - h_2 possibly adds A and has full control over C
- Exercise:
 - If only B is true in the initial state and goal is $A \wedge C$
 - Q: What sequence of HLAs will achieve the goal?



43

NUS Computing

Rational Decision Making

CS4246/CS5446
AI Planning and Decision Making

This lecture will be recorded!

1

Nondeterminism

Partially observable, nondeterministic, and uncertain environments

2

Definition

- Deterministic environment**
 - Next state of the environment is completely determined by current state and action executed

```

graph LR
    S0((S0)) --> A[A]
    A --> S1((S1))
  
```

3

Definition

- Non-deterministic environment**
 - Actions are characterized by different possible outcomes, without any attaching probabilities

```

graph LR
    S0((S0)) --> A[A]
    A -- OR --> S1X((S1-X))
    A -- OR --> S1Y((S1-Y))
    A -- OR --> S1Z((S1-Z))
  
```

4

Non-Determinism

- Non-deterministic effects**
 - Action may have different possible outcomes
- Controllable (angelic) nondeterminism**
 - When an agent itself makes the choices
 - E.g., in hierarchical planning – still deterministic!
- Uncontrollable (demonic) nondeterminism**
 - When an adversary or nature makes the choices
 - Planning in nondeterministic or uncertain environments

RN: 11.4.3

5

Definition

- Stochastic or uncertain environment**
 - Outcomes not fully observable or not deterministic
 - Uncertainty usually characterized in terms of probabilities

```

graph LR
    S0((S0)) --> A[A]
    A -- "0.1" --> S1X((S1-X))
    A -- "0.3" --> S1Y((S1-Y))
    A -- "0.6" --> S1Z((S1-Z))
  
```

6

Some Terminology

- State variables (or fluents)**
 - Factored representations of states whose truth values may change over time
- Belief states**
 - Sets of physical states the agent might be in
 - Example: What are the belief states for binary fluents: Have_Cake \wedge Eaten_Cake?
 - Represent agent's current belief about the possible physical states it might be in
 - Entire belief-state space contains every possible set of physical states.
 - If problem P has N fluents, then there are up to 2^N states in the belief space
- Percepts**
 - Percepts supplied by sensors in acting, but a model of sensors is needed in planning
 - Need to reason about percepts obtained when plan is executed in partially observable environment

7

7

Decision Making under Uncertainty

Uncertain states and action effects

8

8

Rational Decision Making

- Main ideas:**
 - Assume episodic, non-deterministic, partially observable environments
 - Make decisions based on combining **beliefs** and **desires**
 - Choose a strategy for experimentation and action that is logically consistent (cf. right) with:
 - basic judgments about the unknown states or events**
 - basic preferences for consequences**
- Rationality**
 - Assume **limited** resources
 - Make decisions that will maximize profit/gain (**or minimize cost/pain**)
 - Rationality as a product (ends) or a process (means)

9

9

Types of Decision Theory

- Normative decision theory**
 - Describes how ideal, rational agents should behave
- Descriptive decision theory**
 - Describes how actual agents (humans) really behave
- Prescriptive decision theory**
 - Prescribes guidelines for agents to behave rationally

10

10

Solving Decision Problems

- Decision (Planning) Problem or Model**
 - Appropriate abstraction of states, actions, uncertain effects, and goals (wrt costs and values or **preferences**)
- Decision Algorithm**
 - Input: a problem
 - Output: a solution in the form of an **optimal** action sequence
 - Optimal action at each decision or choice point**
- Decision Solution**
 - An action sequence or solution from an initial state to the goal state(s)
 - An optimal solution or action sequence; OR**
 - An optimal policy** that specifies "best" action in each state wrt to costs or values or preferences
 - (Optional) A goal state that satisfies certain properties

11

11

Decision Making under Uncertainty

- Decision (Planning) Model:**
 - Actions:** $a \in A$
 - Uncertain current state:** $s \in S$ with probability of reaching: $P(s)$
 - Transition model** of uncertain action outcome or effects: $P(s' | s, a)$ – probability that action a in state s reaches state s'
 - Outcome** of applying action a :
Result(a) – random variable whose values are outcome states
 - Probability of outcome state** s' , conditioning on that action a is executed: $P(\text{Result}(a) = s') = \sum_s P(s)P(s' | s, a)$
 - Preferences** captured by a **utility function**: $U(s)$ – assigns a single number to express the desirability of a state s

12

12

Axioms of Utility

Constraints on rational preferences

13

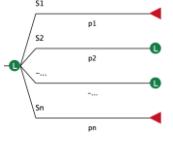
Modeling Preferences

- Preference**
 - For any two acts or events A and B , $A > B$ (A is preferred to B) or $A \sim B$ (A and B are indifferent)
 - Agent is indifferent between two alternatives only if agent:
 - has considered both alternatives; and
 - is completely willing to trade one for the other
- Utility**
 - Classify every item in an individual's preference ordering
 - Applying the conditions of rationality
 - Numeric ranking shows relative importance of an indifference class
 - Relative importance or utility of items in the class
- Utility function or Utility Scale $U(s)$:**
 - Assigns a single number to express desirability of a state
 - Numbering items in a preference ordering

14

Rational Preferences

- Assumptions:**
 - $A > B$: Agent prefers A to B
 - $A \sim B$: Agent is indifferent between A and B
 - $A \geq B$: Agent prefers A over B or is indifferent between them
- Lottery:**
 - Action as ticket to lottery L :
 - with set of outcomes S_1, \dots, S_n
 - with probabilities p_1, \dots, p_n
 - $L = [p_1, S_1; p_2, S_2, \dots, p_n, S_n]$
 - Each outcome S_i of a lottery can be either an atomic state or another lottery



15

Axioms of Utility Theory

- Utility Theory:**
 - Understand how preferences between complex lotteries are related to preferences between the underlying states in the lotteries
- 6 axioms of utility**
 - Constraints on preferences
 - An agent that violates any axiom will exhibit **irrational** behavior in some situations

16

A1: Orderability

- Definition**
 - Given any two lotteries, a rational agent must either:
 - prefer one to the other; OR
 - Rate the two as equally preferable
 - Exactly one of $(A > B)$, $(B > A)$, or $(A \sim B)$ holds
- Example:**
 - A student will prefer
 - taking CS4246 to CS5340
 - taking CS5340 to CS4246
 - Or is indifferent between the two modules

17

A2: Transitivity

- Definition**
 - Given any three lotteries, if an agent prefers A to B and prefers B to C , then the agent must prefer A to C

$$(A > B) \wedge (B > C) \Rightarrow (A > C)$$

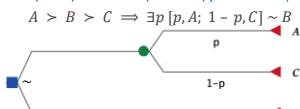
Coke	>	Sprite	
	Coke	>	Pepsi
Sprite	>	Pepsi	

18

A3: Continuity

- Definition

- If some lottery B is between A and C in preference, then there is some probability p for which the rational agent will be indifferent between:
 - Betting B for sure; AND
 - The lottery that yields A with probability p and C with probability $1 - p$

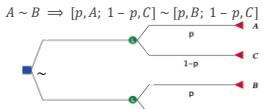


19

A4: Substitutability

- Definition

- If an agent is indifferent between two lotteries A and B , then the agent is indifferent between two more complex lotteries that are the same except that B is substituted for A in one of them
- This holds regardless of probabilities and other outcome(s) in lotteries
- This also holds if \succ is substituted for \sim in the axiom



20

19

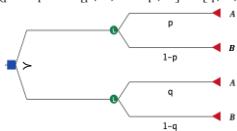
20

A5: Monotonicity

- Definition

- Suppose two lotteries have the same two possible outcomes, A and B .
- If an agent prefers A to B , then the agent must prefer the lottery that has a higher probability for A (and vice versa)

$$A > B \Rightarrow (p > q \Leftrightarrow [p, A; 1 - p, B] > [q, A; 1 - q, B])$$



21

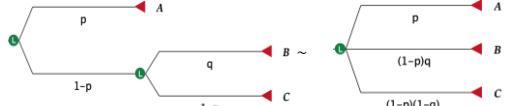
21

A6: Decomposability

- Definition

- Compound lotteries can be reduced to simpler ones
- No fun in gambling rule – two consecutive lotteries can be compressed into a single equivalent lottery

$$[p, A; 1 - p, [q, B; 1 - q, C]] \sim [p, A; (1 - p)q, B; (1 - p)(1 - q), C]$$



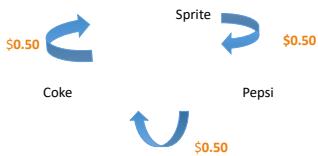
22

22

Exploiting Irrationality

- E.g., Coke > Sprite > Pepsi > Coke

- How to make money from someone without the above preference structure?



23

23

Rational Preferences and Utility

- Note:

- Axioms of utility are really axioms about preferences
- Nothing is mentioned about a utility function
- Need to derive consequences from axioms of utility

- How should a rational agent behave?

- If an individual can satisfy the conditions of rationality, then, a utility function U can be constructed according to:
 - Von Neumann and Morgenstern (1944) Expected Utility Theorem

24

24

Expected Utility Theorem

Von Neumann and Morgenstern (1944)

25

Existence of Utility Function

- If an agent's preferences obey the axioms of utility; then there exists a function U such that:
 - $U(A) > U(B)$ if and only if A is preferred to B , and
 - $U(A) = U(B)$ if and only if the agent is indifferent between A and B .

$$U(A) > U(B) \Leftrightarrow A > B \quad \text{and} \quad U(A) = U(B) \Leftrightarrow A \sim B$$

26

Expected Utility of A Lottery

- The utility of a lottery is the sum of the probability of each outcome times the utility of that outcome:

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

- Once the probabilities and utilities of the possible outcome states are specified, the utility of a compound lottery involving those states is completely determined.

$$U(L) = pU(A) + (1-p)U(B)$$

27

Computing Expected Utility

- Expected utility**
 - Note that outcome of a nondeterministic action is a lottery
 - Expected utility of an action a is the average utility value of the outcomes, weighted by the probability that the outcome occurs
$$EU(a) = \sum_{s'} P(\text{Result}(a) = s')U(s') = \sum_{s'} \sum_s P(s)P(s'|s, a)U(s')$$
- Non-unique utility functions**
 - Agent's behavior doesn't change if U is subjected to an **affine transformation**:
$$U'(s) = aU(s) + b \text{ with } a > 0$$

28

Maximum Expected Utility Principle

Basic principle of decision theory

29

Rational Decision Making

- An agent can act **rationally** – consistently with its preferences – only by choosing an action that maximizes expected utility according to:
- Maximum expected utility (MEU) principle**
 - A rational agent should choose the action that maximizes its **expected utility**

$$\text{action} = \underset{a}{\operatorname{argmax}} EU(a)$$

- A prescription for intelligence behavior – “do the right thing” – a basis for AI

30

<h2>Rational Decision-Theoretic Agent</h2> <ul style="list-style-type: none"> • Preference assumption <ul style="list-style-type: none"> ▪ Assumes that agent's preferences satisfy all the axioms or conditions of rationality • Rationality assumption <ul style="list-style-type: none"> ▪ Assumes that agent always act rationally by choosing the most preferred option • Ideal agent assumption <ul style="list-style-type: none"> ▪ Agents of the theorem are ideal and hypothetical beings, but can be used as guides for our own problem solving or decision making • Other assumptions: <ul style="list-style-type: none"> ▪ Existence of a utility function that describes an agent's preference behavior does not necessarily mean that agent is explicitly maximizing that utility function itself ▪ An observer can learn about the utility function that represents what the agent is actually trying to achieve through the agent's behavior <p style="text-align: center;">31</p>	<h2>MEU as Performance Measures</h2> <ul style="list-style-type: none"> • Given a set of environments, agents, and relevant perception histories: <ul style="list-style-type: none"> ▪ If an agent acts so as to maximize a utility function that correctly reflects the performance measure, then the agent will achieve the highest possible performance score (averaged over all the possible environments). • Main ideas: <ul style="list-style-type: none"> ▪ Transition from external performance measure to internal utility function ▪ Performance measure: <ul style="list-style-type: none"> ▪ Gives a score for a history—a sequence of states ▪ Applied retrospectively after an agent completes a sequence of actions ▪ Utility function can be used to guide actions step by step <p style="text-align: center;">32</p>
<h2>Computing MEU</h2> <ul style="list-style-type: none"> • What are the major challenges? <ul style="list-style-type: none"> ▪ Considering many actions a ▪ Estimating $P(s)$ over possible states of the world in $P(\text{RESULT}(a) = s')$ requires perception, learning, knowledge representation, and inference ▪ Computing $P(\text{RESULT}(a) = s')$ requires a causal model of the world ▪ Computing outcome utilities $U(s')$ requires further searching or planning ▪ Estimating uncertainty about U • In summary, decision theory: <ul style="list-style-type: none"> ▪ Provides a basic, general mathematical framework to define the AI problem ▪ Cannot solve the AI problem! <p style="text-align: center;">33</p>	<h2>Utility Functions</h2> <p>Basic Concepts and Assessment Methods</p> <p style="text-align: center;">34</p>
<h2>Encoding Preferences in Utility Functions</h2> <ul style="list-style-type: none"> • Assess utility or value functions <ul style="list-style-type: none"> ▪ To measure "desirability" of different outcomes and trade-off situations • Possible scales: <ul style="list-style-type: none"> ▪ Monetary cost ▪ Revenue, profit ▪ Life expectancy ▪ Jobs saved ▪ etc. <p style="text-align: center;">35</p>	<h2>Measuring Utilities</h2> <ul style="list-style-type: none"> • Provide summary scores that <ul style="list-style-type: none"> ▪ aggregate different aspects of "goodness" measures ▪ incorporate attitudes towards risks and "goodness" quantities • Assessment methods <ul style="list-style-type: none"> ▪ By direct assessment using: <ul style="list-style-type: none"> ▪ probability equivalent ▪ certainty equivalent ▪ By consensus ▪ From published reports • Multi-attribute utility functions may be used <p style="text-align: center;">36</p>

Utility Functions

- A utility function, U , represents a way to translate dollars or other "desirability" measures, x , into utility units $U(x)$.

Graphical Representation

Tabular Representation

Wealth (x)	Utility Value $U(x)$
2500	1.50
1500	1.24
1000	0.93
600	0.65
400	0.47
0	0.15

Mathematical Representation

$$U(x) = \log(x)$$

$$U(x) = 1 - e^{-x/R}$$

$$U(x) = x^{0.5}$$

37

Preference Elicitation: Method 1

- Using probability equivalents**
 - Present choices to agents; define the utility function from observed responses
- Method:**
 - Fix a scale; fix utilities of 2 outcomes to establish the scale
 - Worst: u_L ; Best: u_T
 - Normalized utilities: $u_L = 0; u_T = 1$
 - Assess utility for prize S by asking the agent to choose between $U(S)$ and the lottery $[p, u_T; (1-p), u_L]$
 - Adjust p until the agent is indifferent between S and lottery

38

Example: Getting Good Grade

- Demonstration:**
 - Grade in CS4246/CS5446: $U(F) = 0; U(A) = 1$; sure grade: B, C, D
 - What value of p would you trade A for B ?

39

Certainty Equivalent

- Certainty Equivalent (CE)**
 - Fixed amount of money equivalent to a given situation involving uncertainty
 - Value agent accepts in lieu of the lottery
- Example:**
 - You are faced with the following lottery:

Win	\$2000	with probability	0.5
Lose	\$20	with probability	0.5

 EMV = \$990
 - How much are you willing to sell the lottery for?
 - This amount, say $\$X$, is the least that you would accept for the lottery, then the lottery must be **equivalent** in your mind to a sure $\$X$

40

Preference Elicitation: Method 2

- Using certainty equivalents
- Determine two initial points on curve
- Arbitrarily assign utility values to the points
- Create reference lottery to determine CE
- Find third point on curve by formula:
 - $U(CE_1) = EU(\text{Lottery})$
- Create additional reference lotteries
- Proceed as before until enough points are available to plot a curve

Example:

- Assume: worst: $\$10$, best: $\$100$
- Set $U_L(10) = 0; U_T(100) = 1$
- Reference lottery
- $U(CE_1) = EU(\text{Lottery})$
- $U(CE_1) = U(30)$
 $= 0.5U(100) + 0.5U(10)$
 $= 0.5$

41

Continue ...

CE_2

$$U(50) = 0.5U(100) + 0.5U(30) = 0.75$$

CE_3

$$U(18) = 0.5U(30) + 0.5U(10) = 0.25$$

42

Plotting the Utility Function

43

Example: Micromort

value that people place on their own lives

- Micromort:**
 - A one in a million chance of death, often used as a unit of risk
 - People appear to be willing to pay about \$50 per micromort, e.g., \$10,000 for a safer car that halves the risk of death (from 400 to 200 micromorts).
 - Only for small risks, most people won't kill themselves for \$50 million.
 - See video: <https://www.youtube.com/watch?v=G3wolqD-acQ>

44

Example: Quality Adjusted Life Year

- QALY:**
 - A scale commonly used in health care literature
 - One year in perfect health is 1 QALY
 - One year bedridden would be less preferred, e.g. 0.5 QALY.
 - Death is 0 QALY
 - See video: <https://www.youtube.com/watch?v=3tDXwKVkn68>
 - Whose utility values are these?

45

Utility of Money and Risk Attitudes

46

Expected Monetary Value

- Assumption:**
 - Agent prefers more money than less – monotonic preference
- Example:**
 - You have won \$1 mil so far
 - Now, flip a coin – Lose all the money (\$0) if Head and gain \$2.5mil if Tail
 - Will you play?
- What is the expected monetary value (EMV) of the lottery?**
 - $EMV = [0.5, 0; 0.5, 2.5 \text{ mil}] = \1.25 mil
- Is accepting the game the best decision?**
 - Yes according to MEU Principle – EMV is higher than \$1 mil won so far!

47

Utility of Money

Risk averse
Risk seeking

A typical utility of money function

Source: RN16.2 b

48

- Let S_n denote the state of having $\$n$; current wealth is $\$k$
 - $EU(\text{Accept}) = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+2.5\text{mil}})$
 - $EU(\text{Decline}) = U(S_{k+1\text{mil}})$
- To decide:
 - Assign utility values to the outcome states:
 $S_k, S_{k+2.5\text{mil}}, S_{k+1\text{mil}}$
 - Will you accept the lottery if:
 $S_k = 5, S_{k+2.5\text{mil}} = 9, S_{k+1\text{mil}} = 8$
 - $EU(\text{Accept}) = 0.5(5) + 0.5(9) = 7$
 - $EU(\text{Decline}) = 8$
- Note:
 - Utility of money is usually proportional to log of the amount

Limitations of the EMV criterion

- Using money as decision objective:

- Intuitive only if objective can be measured in terms of monetary value
- Considers only the average or expected value; ignores the range of possible values
- Does not take into account **risk attitudes**

49

Risk Attitudes

- A patient is forced to play the following game:

- Live for another 30 years with probability 0.5
- Die immediately with probability 0.5

- Will he choose to live just for 5 years to get out of the game?

- Risk behaviors:

- If he would trade a lottery for a sure amount that is less than the expected value, he is **risk-averse**
- If he would pay an amount more than the expected value to enter the lottery, then he is **risk-seeking**
- Otherwise, he is **risk-neutral**

50

Risk or Insurance Premium

- Risk attitudes and utility functions



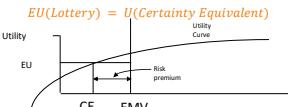
- Risk Premium

- Risk premium** = EMV - Certainty Equivalent
- Premium paid, in the sense of a lost opportunity, to avoid the risk
- Risk premium and behavior**
 - For **risk-averse** individual, risk premium is positive
 - For **risk-seeking** individual, risk premium is negative
 - For **risk-neutral** individual, risk premium is zero

51

Expected Utility and Certainty Equivalent

- Expected utility of a lottery is equal to the utility of its certainty equivalent:



- Note

- If two alternatives have the same CE, then they must have the same EU; the decision maker would be indifferent to a choice between the two
- Ranking alternatives by their CEs is the same as ranking them by their expected utilities

52

Example: Paying to Avoid Risk

- Recall the following gamble:

- Win \$2000 with probability 0.5
- Lose \$20 with probability 0.5

- What is the EMV?

- EMV = \$990

- How much will you be willing to accept to avoid the lottery?

- Let CE = \$300

- What is the risk premium?

- Risk premium = \$990 - \$300 = \$690

- What does this mean?

- Trading lottery for \$300: willing to give up \$690 in expected value to **avoid risk**

53

Calculating Risk Premium

- Recall:

- Risk premium = EMV - Certainty Equivalent
- $EU(\text{Lottery}) = U(\text{Certainty Equivalent})$

- How to calculate risk premium given a lottery and a utility function?

- Find EU for the lottery
- Find CE, amount with utility value given by EU
- Calculate EMV for the lottery
- Risk premium = EMV - CE

- Other methods: Utility functions for non-monetary attributes

- Utility functions derived for attributes other than money
- Set up arbitrary scale with extremes (highest and lowest utilities)
- Use either CE or PE assessment methods

54

54

Some Caveats

- Utilities do not add up
 - $U(a + b) \neq U(a) + U(b)$
 - Utility functions are non-linear
 - Calculate net payoffs at end-points of decision tree before transforming to utility values
- Utility differences do not express strengths of preferences
 - Utility provides numerical scale for ordering preferences, not a measure of their strengths
- Utility function
 - A subjective personal statement of an individual's preferences
 - Provides no basis for comparing utilities among individuals
- Certainty equivalent vs expected utility
 - Certainty equivalent is measured in \$ (or any basic unit)
 - Expected utility is measured on the utility scale

55

 NUS | Computing

Decision Analysis

CS4246/CS5446
AI Planning and Decision Making

1

Types of Decision Theory

- **Normative decision theory**
 - Describes how ideal, rational agents should behave
- **Descriptive decision theory**
 - Describes how actual agents (humans) really behave
- **Prescriptive decision theory**
 - Prescribes guidelines for agents to behave rationally

2

Recall: Solving Decision Problems

- **Decision Problem or Model**
 - Appropriate abstraction of states, actions, **uncertain** effects, and goals (wrt costs and values or preferences)
- **Decision Algorithm**
 - Input: a problem
 - Output: a solution in the form of an action sequence
 - Optimal action at each decision or choice point
- **Decision Solution**
 - An action sequence or solution from an initial state to the goal state(s)
 - **An optional solution or action sequence**; OR
 - An optimal policy that specifies "best" action in each state wrt to costs or values or preferences
 - (Optional) A goal state that satisfies certain properties

3

Recall: Decision Making under Uncertainty

- **Decision Model:**
 - **Actions:** $a \in A$
 - **Uncertain current state:** $s \in S$ with probability of reaching: $P(s)$
 - **Transition model** of uncertain action outcome or effects:
 $P(s'|s, a)$ – probability that action a in state s reaches state s'
 - **Outcome** of applying action a :
 $\text{Result}(a)$ – random variable whose values are outcome states
 - **Probability of outcome state** s' , conditioning on that action a is executed:
 $P(\text{Result}(a) = s') = \sum_s P(s)P(s'|s, a)$
 - **Preferences** captured by a **utility function**:
 $U(s)$ – assigns a single number to express the desirability of a state s

4

Decision Analysis

A prescriptive framework for decision making

5

What is Decision Analysis?

- **Emerged in the 1960s from operations research and game theory**
 - (Howard, 1966)
 - (Raiffa and Abbas, 2016)
- **A prescriptive framework**
 - Vs. Normative decision theory - Assumes decision makers as ideally rational agents
 - Vs. Descriptive decision theory - Describes how people actually make decisions
- **Provides structure and guidance for thinking systematically and recommends alternatives about hard decisions**
 - Can help make better decisions, but not improve luck nor guarantee good outcomes!
- **Why does it matter for AI Planning and Decision Making?**

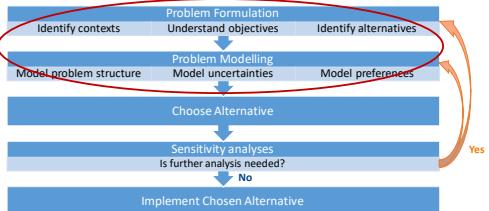
6

How to Improve Decision Making?

- **Complexity Management**
 - Effective methods for problem organization
 - Vocabulary on elements of decision structure
 - Graphical structuring tools
 - Solution and analysis procedures
- **Uncertainty Reduction**
 - Identification of sources of uncertainty
 - Quantitative representation of uncertainty
- **Multiple Objectives Specification**
 - A framework and specific tools for dealing with multiple objectives
- **Multiple Perspectives Resolution**
 - A framework and specific tools to sort through and resolve differences

7

The Decision Analysis Process



8

Example: Cryptocurrency Investment Problem

- **Decision:**
 - Whether or not to invest in a new cryptocurrency – KittyCoin
- **Objective:**
- **Considerations:**
 - Technology and exchange platforms have great credentials
 - Proposed project is more risky than most other cryptocurrencies
- **Possible consequences:**
 - If invest and value of KittyCoin rises, there will be high monetary returns
 - If not, capital may be put in stock market or other investment options

9

Problem Formulation

- **Identify the decision context**
 - Define precise problem to solve
 - Identify the decision maker and perspective
- **Identify objectives**
 - What are the goals of the decision?
- **Identify alternatives or actions**
 - Careful inspection of all aspects of a problem can lead to discovery of new alternatives
- **Using:**
 - Fundamental objectives
 - Other objectives (e.g., means objectives)

10

Organizing Objectives

- **Structure or organize objectives to**
 - reveal all relevant details to be achieved
 - allow direct incorporation into problem definition
- **Fundamental Objectives**
 - Reflect things that the problem solver needs to achieve
 - Organized into hierarchies
- **Means Objectives**
 - Help achieve other objectives
 - Organized into networks

11

Example: Cryptocurrency Investment Problem

-
- ```

graph TD
 A[Maximize Satisfaction] --> B[Minimize Energy Consumption]
 A --> C[Minimize Time Spent]
 A --> D[Maximize Return ($)]
 B --> E[Research]
 C --> F[Transactions]

```
- **How to construct the fundamental objectives hierarchy?**
    - Ultimately, for any objective: Why is it important?
    - To move downward: What do you mean by that?
    - To move upward: Of what more general objective is this an aspect?
  - **The lowest level form the basis to measure consequences**
  - **Identify alternatives to achieve objectives**

12

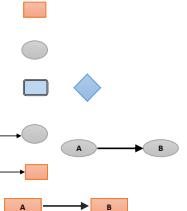
## Decision Model Formulation

- Use “divide and conquer” method
  - To understand its structure
  - To measure the uncertainty, and
  - To specify the preferences
- **Modeling Structure**
  - Construct graphical, mathematical model to denote structure of decision problem
  - Influence Diagram
  - Decision Tree

13

## Basic Elements of a Decision Model

- **Decision nodes**
  - Decision points with several choices or alternatives
- **Chance nodes**
  - Uncertain or chance events with several outcomes
- **Value or utility nodes**
  - Deterministic monetary value or utility functions
  - Measure desirability of final objectives
- **Probabilistic dependencies**
  - Represent conditional dependence of chance events
- **Informational dependencies**
  - Represent information available at decision points



14

## Relevance and Sequence

- **Relevance Arcs**
  - Predecessor is relevant for assessing the chances associated with the uncertain event or the values of the value (consequence) node
- **Sequence Arcs**
  - Decision is made knowing the outcome of the predecessor node
  - The decision maker knows the outcome of event A when making decision B
  - Decision A is made before decision B

15

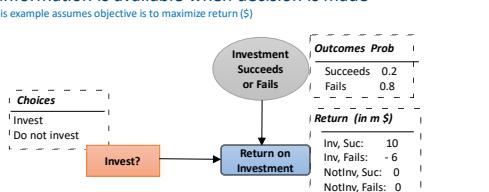
## Decision Networks

Influence Diagrams

16

## The Basic Risky Decision

- No information is available when decision is made
  - This example assumes objective is to maximize return (\$)



| Outcomes Prob |     |
|---------------|-----|
| Succeeds      | 0.2 |
| Fails         | 0.8 |

| Return (in m \$) |    |
|------------------|----|
| Inv, Suc:        | 10 |
| Inv, Fails:      | -6 |
| Notinv, Suc:     | 0  |
| Notinv, Fails:   | 0  |

Example: Cryptocurrency Investment Problem

17

## Building Influence Diagrams

- Hierarchical representation to facilitate communication
  - Captures current state of knowledge of decision maker
  - Details hidden beneath the structure to favor simplicity
- **No best strategy for building influence diagrams; best approach is:**
  - Put together a simple version of diagram
  - Add details as necessary until all relevant aspects are included
- **No fixed order of “reasoning”**
- **Some Common Mistakes:**
  - Influence diagrams (like BNs) are not flowcharts
    - An influence diagram is a snapshot of the decision situation
    - An arrow from a chance node to a decision node means that the chance event outcome is known at the time of decision
  - **No cycles** are allowed in an influence diagram

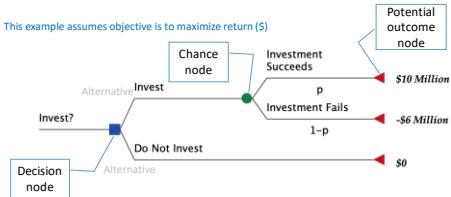
18

## Decision Trees

19

## The Basic Risky Decision

- This example assumes objective is to maximize return (\$)



Example: Cryptocurrency Investment Problem

DT and DN are isomorphic

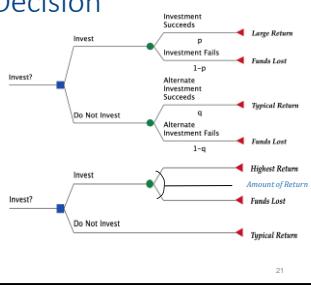
DT can potentially give immediate insights

20

## The Basic Risky Decision

### Double Risky Decision

- e.g., if the investor decides not to invest, she may lose the money in the stock market or the less risky investment



21

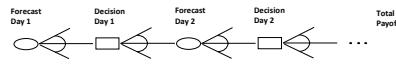
21

## Sequential Decisions

- Number of branches increase exponentially as number of decisions and chance events increase

- May use “skeleton” version of decision tree to represent sequential decisions if:

- the sequential problem repeats itself; and
- the decision tree is symmetric at each stage



22

## Building Decision Trees

- Start with the first (leftmost) node:
  - For decision node, draw a branch for each alternative
  - For chance node, draw a branch for each possible outcome
- Label each branch emanating from a chance node with:
  - a unique outcome
  - the corresponding path dependent probability, and
  - the path dependent value for that branch
- Label each branch emanating from a decision node with:
  - a unique alternative, and
  - the value for that alternative
- Place final values on terminal nodes at the end of each path

23

23

## Correspondence between ID and DT



Outcomes of event B are probabilistically dependent on outcomes of event A



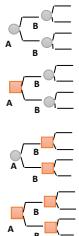
Outcomes of event B are probabilistically dependent on choices of decision A



The decision maker knows the outcome of event A when making decision B



Decision A is made before decision B



24

24

## Modeling Uncertainty

- Apply probability theory to represent uncertainty

- Approaches:
  - Using subjective judgment in assessing probabilities – difficult!
  - Using theoretical probability models
  - Using data

25

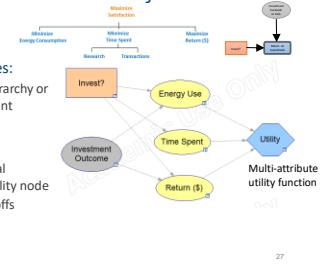
## Modeling Preferences

- Use utility functions
- Assess utility or value functions to measure “desirability” of different outcomes and trade-off situations
- Possible scales:
  - Monetary cost
  - Revenue, profit
  - Life expectancy
  - etc.

26

## Representing Fundamental Objectives

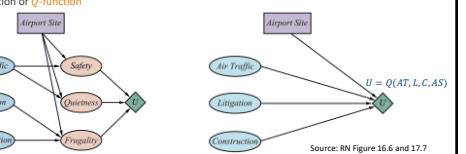
- In decision networks:
- To represent multiple attributes:
  - Use fundamental objectives hierarchy or utility node structure to represent multiple attributes
- To capture trade-offs:
  - Aggregate function for individual attribute utilities in “overall” utility node
  - Incorporate appropriate trade-offs among the different objectives



27

## Example: Multi-attribute Utility Function

- Utility node:
  - Represents the utility function; parents are all the variables (attributes) that directly affect utility
  - A special kind of deterministic nodes - outcome is certain given values of the parents
  - Simplified form: Utility node represents expected utility associated with each action: the action-utility function or Q-function

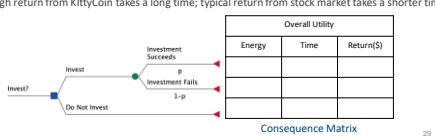


Source: RN Figure 16.6 and 17.7

28

## Representing Fundamental Objectives

- In decision trees:
- Normally assign only one final value to each terminal node outcome or branch in consequence matrix
  - Collapse multidimensional objective description into single score for final consequence
  - Require assessment and application of appropriate trade-off weight
    - e.g., High return from KittyCoin takes a long time; typical return from stock market takes a shorter time



29

## Solving a Decision Model

- $$EU(a) = \sum_s P(\text{Result}(a) = s')U(s') = \sum_a \sum_s P(s|a)P(s'|s, a)U(s')$$
- Solution algorithms often find optimal decisions by:
    - Chance values expectation
    - Decision value maximization
  - Common solution methods for decision trees:
    - Rollback algorithm
    - Monte Carlo simulation
  - Common solution methods for influence diagrams:
    - Reduction algorithm
    - Sampling
    - Inference in Bayesian networks

30

30

## Chance Values Expectation

- Determine expected value for a chance node
  - e.g. Consider a chance event "Investment Outcome" with possible outcomes "Gain," "Unchanged," "Loss"

| Investment Outcome | Outcome   | Probability | Value (utility) |
|--------------------|-----------|-------------|-----------------|
|                    | Gain      | 0.2         | 1.0             |
|                    | Unchanged | 0.3         | 0.5             |
|                    | Loss      | 0.5         | 0               |

$\bullet \text{EU}[\text{Investment outcome}] = 0.2(1.0) + 0.3(0.5) + 0.5(0.0) = 0.35$

31

## Decision Values Maximization

- Determine maximum value for a decision node
  - e.g. Consider a decision point "Invest?" with possible alternatives "Invest" and "Not Invest"

| Alternative | Value (\$ in M) |
|-------------|-----------------|
| Invest      | 6               |
| Not Invest  | 1               |

$\bullet \text{Max}[\text{Invest?}] = \text{Max}(6, 1) = 6$

32

## Solving an Influence Diagram

- Solution obtained by reducing diagram, while recording optimal choices, into single value node
- Final value node contains expected maximum value achievable with optimal decisions

33

## Solving an Influence Diagram: Common Approach

- Set evidence variable to the observed state
- For each possible value of the decision node:
  - Set the decision node to that value
  - Calculate the posterior probability of the parents of the utility using probabilistic inference algorithm
  - Calculate the resulting utility for the action
- Return the action with the highest utility

Use efficient Bayesian Network inference algorithms

34

## Solving a Decision Tree (Rollback)

35

## Solving a Decision Tree

- Choose among risky alternatives
  - Pick alternative with the highest EMV or utility
- Folding or rolling back the tree:
  - Begin at end points of branches on far right-hand side and move to left
  - Chance value expectation
    - Calculate expected values when encountering a chance node; OR
  - Decision value maximization
    - Choose the branch with the highest value or expected value when encountering a decision node
  - Record the value accumulated and alternative selected for each decision node along the optimal path

36

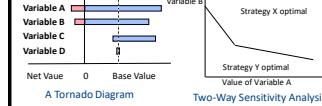
## Sensitivity Analysis

- Answers the question:  
“What if ... is changed in this decision?”
- Determines what matters and what does not throughout the modeling process
- Provides guidance to develop requisite decision model
- Integral part of decision analysis, may lead to:
  - Problem re-definition (what if something changed?)
  - New objectives, alternatives, structure, uncertainty, preference
- No optimal procedures, useful techniques facilitate :
  - identifying and structuring problems
  - detecting dominance among alternatives
  - assessing probabilities and utilities

37

## Sensitivity Analysis Methods

- Common Methods:**
  - Structural analysis methods
  - The clarity test
  - Tornado diagrams
  - One, two, three-way analysis
- The Clarity Test:**
  - Resolve ambiguities
- Definition:**
  - Imagine a clairvoyant who has access to all future information
  - Would the clairvoyant be able to determine unequivocally what the outcome would be for every event in the decision model?
  - No interpretation or judgment should be required of the clairvoyant
- Example:**
  - Does “Temperature” of “high, medium, or low” pass the clarity test?



38

## Robust Decisions

- Robust or minimax decision – Extreme case
  - One that gives the best result in the worst case.
  - “Worst case” means worst with respect to all plausible variations in the parameter values of the model
  - Letting  $\theta$  stand for all the parameters in the model, the robust decision is defined by:  

$$a^* = \max_a \min_{\theta} EU(a; \theta)$$
- For parametric uncertainty
  - Bayesian decision theory: model with hyperparameters
  - Analyze performance of hyperparameters or ranges of parameters
- For structural uncertainty
  - Examine ensemble of models
  - Use domain knowledge to propose alternatives

39

## Requisite Decision Model

- When no new intuition emerges about the problem
- When it contains everything essential for solving the problem
- The decision maker’s**
  - thoughts about the problem
  - beliefs regarding uncertainty and preferences**are fully developed**

40

## Value of Information

41

## Information and Decision Making

- Information is gathered to reduce uncertainty in decision making**
  - Consult experts
  - Conduct surveys
  - Perform mathematical or statistical analyses
  - Do research
  - Read books, journals, newspapers
  - Learn from past data
- Relevant questions:**
  - How to evaluate or measure the “value” or usefulness of the information?
  - What does it mean for a knowledge source to provide perfect information?
  - Shall we invest effort or pay \$X for additional information to help problem solving?

42

## Value of Information

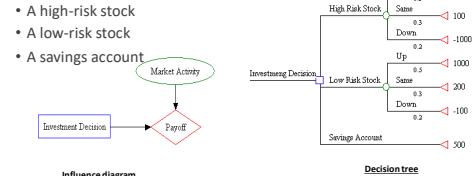
- Costly information gathered to reduce uncertainty
  - How to place value on information in a problem?
  - How to decide whether or not to gather more information?
- Main ideas:
  - Information has value to the extent that it is likely to cause a change of plan and to the extent that the new plan is significantly better than the old one
  - Use conditional probabilities and Bayes' Theorem to model the expected value of information

43

43

## Example: Stock Investment

- An investor may invest in one of three choices:



Decision tree

44

44

## Expected Value of Information

- Expected value of information (EVI)
  - Indicates if information is worth gathering
  - Lower bound: zero expected value
  - Upper bound: expected value of perfect information
- Information has:
  - no value or zero expected value if the same choice will be made before and after obtaining information
  - positive expected value if it leads to a different choice
  - maximum expected value if information is perfect
- EVI is defined in terms of the decision context
  - Different people in different situations may place different values on the same information

45

45

## Stock Investment Example (cont.)

Should investor consult an expert?

- If the expert always provides perfect information:
  - $P(\text{Exp says "Up"} | \text{Market Up}) = 1$
  - $P(\text{Exp says "Down"} | \text{Market Up}) = 0$
  - $P(\text{Exp says "Up"} | \text{Market Down}) = 0$
- To show there is no uncertainty after hearing the expert, apply Bayes' theorem:  

$$P(\text{Market Up} | \text{Exp says "Up"}) = \frac{P(\text{Exp says "Up"} | \text{Market Up}) P(\text{Market Up})}{P(\text{Exp says "Up"} | \text{Market Up}) P(\text{Market Up}) + P(\text{Exp says "Up"} | \text{Market Down}) P(\text{Market Down})}$$

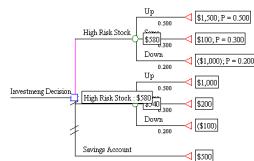
$$= \frac{1}{1} = 1$$
- Note:
  - The posterior probability is equal to 1 regardless of the prior probability
  - How do real problems differ from the above situation?

46

46

## Stock Investment Example (cont.)

- The optimal choice is the high-risk stock with EMV \$580
- Assumption: optimistic about market (Up, 0.5 prob)
- How much would the investor be willing to pay for information about the market activity?



47

47

## Expected Value of Perfect Information

- Max amount that the decision maker is willing to pay for perfect information

- To find expected value of perfect information (EVPI):
  - Modify the decision model to indicate perfect information
  - Solve the model and find the EMV (\$1000)

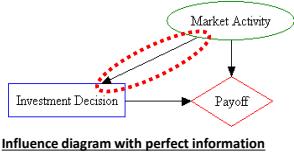
$$\begin{aligned} \bullet \text{EVPI} &= \text{EMV}(\text{with perfect information}) - \text{EMV}(\text{original}) \\ &= \$1000 - \$580 = \$420 \end{aligned}$$

48

48

## Modifying Influence Diagram for EVPI

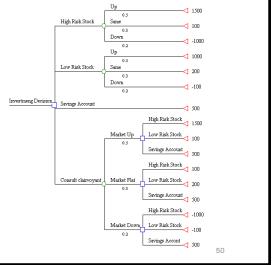
- Impose order on the decision and uncertain event nodes
- The uncertainty nodes for which perfect information is available comes before the decision node



49

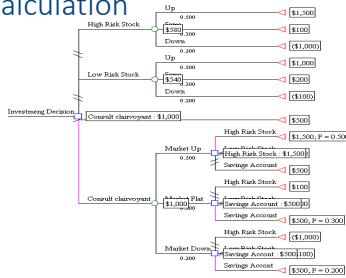
## Modifying Decision Tree for EVPI

- Reorder the decision and the uncertain event nodes
- The uncertainty nodes for which perfect information is available comes before the decision node



50

## EVPI Calculation



51

## (Expected) Value of Perfect Information

Assume exact evidence about variable  $E_j$ ; compute value of perfect information (VPI)

- Given expected utility with current best action  $\alpha$ :

$$EU(\alpha) = \max_a \sum_{s'} P(Result(a) = s')U(s')$$

- Value of the best new action after  $E_j = e_j$  is obtained

$$EU(\alpha_{e_j}|e_j) = \max_a \sum_{s'} P(Result(a) = s'|e_j)U(s')$$

- Variable  $E_j$  can take multiple values  $e_j$ , so on averaging:

$$VPI(E_j) = \sum_{e_j} P(E_j = e_j)EU(\alpha_{e_j}|E_j = e_j) - EU(\alpha)$$

52

## Properties of Value of information

- Expected value of information is always non-negative  
 $\forall j \quad VPI(E_j) \geq 0$
- VPI is not additive  
 $VPI(E_j, E_k) \neq VPI(E_j) + VPI(E_k)$
- VPI is order independent  
 $VPI(E_j, E_k) = VPI(E_j) + VPI(E_k|E_j) = VPI(E_k) + VPI(E_j|E_k) = VPI(E_k, E_j)$

53

## Information Gathering, Decision-Theoretic Agent

- Agent should gather information before taking actions, if possible
  - Cost associated with getting the information, so how to choose which variable to get more information about?

```
function INFORMATION-GATHERING-AGENT(percept) returns an action
 persistent: D, a decision network
 integrate percept into D
 j ← the value that maximizes $VPI(E_j) / C(E_j)$
 if $VPI(E_j) > C(E_j)$
 then return Request(Ej)
 else return the best action from D
```

Unit gain per unit cost  
OR  
Information value

Source: RN Figure 15.9

54

53

54

 NUS Computing

## Markov Decision Process

CS4246/CS5446  
AI Planning and Decision Making

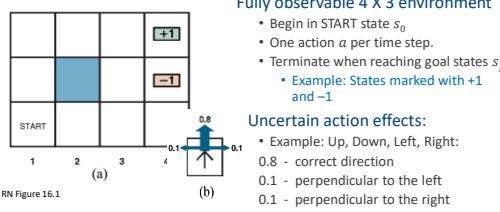
1

## Model Formulation

Define the problem elements

2

### Example: Navigation in Grid World



Source: RN Figure 16.1

**Fully observable 4 X 3 environment**

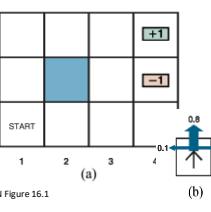
- Begin in START state  $s_0$
- One action  $a$  per time step.
- Terminate when reaching goal states  $s_g$ 
  - Example: States marked with +1 and -1

**Uncertain action effects:**

- Example: Up, Down, Left, Right:
  - 0.8 - correct direction
  - 0.1 - perpendicular to the left
  - 0.1 - perpendicular to the right

3

### Example: Navigation



Source: RN Figure 16.1

**Transition model:**

- Define (stochastic) outcome of each action  $a$
- $P(s'|s, a)$  – probability of reaching state  $s'$  if action  $a$  is done in state  $s$ 
  - Also denoted as  $T(s, a, s')$

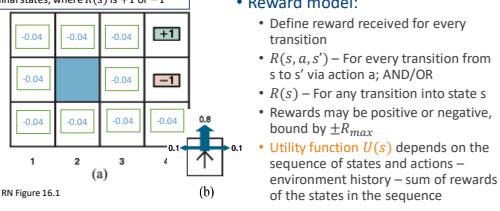
**Markovian assumption:**

- Probability depends only on state  $s$  and not history of earlier states.

4

### Example: Navigation

E.g.,  $R(s) = -0.04$  for all states, except the terminal states, where  $R(s)$  is +1 or -1



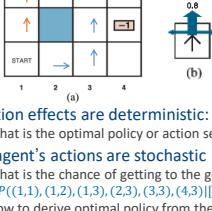
Source: RN Figure 16.1

**Reward model:**

- Define reward received for every transition
- $R(s, a, s')$  – For every transition from  $s$  to  $s'$  via action  $a$ ; AND/OR
- $R(s)$  – For any transition into state  $s$
- Rewards may be positive or negative, bound by  $\pm R_{max}$
- Utility function  $U(s)$  depends on the sequence of states and actions – environment history – sum of rewards of the states in the sequence

5

### Example: Navigation



Source: RN Figure 16.1

**Uncertain action effects:**

- Up, Down, Left, Right:
  - 0.8 - correct direction
  - 0.1 - perpendicular to the left
  - 0.1 - perpendicular to the right

**If action effects are deterministic:**

- What is the optimal policy or action sequence?

**But agent's actions are stochastic**

- What is the chance of getting to the goal with:  $[U, U, R, R, R]$ ?  
 $P((1,1), (1,2), (1,3), (2,3), (3,3), (4,3))|U, U, R, R, R] = 0.8^5 = 0.32768$ .
- How to derive optimal policy from the transition function directly?

6

## Markov Decision Process (MDP)

- **Formally:**

- An MDP  $M \triangleq (S, A, T, R, \gamma)$  consists of:
  - A set  $S$  of states
  - A set  $A$  of actions
  - A transition function  $T: S \times A \times S \rightarrow [0,1]$  that satisfies the **Markov property** such that:
 
$$\forall s \in S, \forall a \in A: \sum_{s' \in S} T(s, a, s') = \sum_{s' \in S} P(s'|s, a) = 1$$
  - A reward function  $R: S \rightarrow \mathbb{R}$  or  $R: S \times A \times S \rightarrow \mathbb{R}$
  - A discount factor  $0 < \gamma < 1$
  - Solution is a **policy** – a function to recommend an action in each state:  $\pi: S \rightarrow A$ 
    - Solution involves careful balancing of risk and reward

7

## Transition Function

- **Formally:**

- $T(s, a, s') = P(s'|s, a)$  is the probability of going from state  $s$  to state  $s'$  taking action  $a$
- Define  $T(s, a, s')$  for all  $s, s' \in S, a \in A$

- **Markov Property**

- The next state is conditionally independent of the past states and actions given the current state  $s$  and action  $a$ , i.e.,
 
$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = P(s_{t+1}|s_t, a_t)$$
 for all  $s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0$

8

## Markov Property

- Is the Markov Property applicable in real-world problems?
  - It is a simplifying assumption!
- Potential violations of Markov assumption
  - State variables and dynamics of environment not captured in model
  - Inaccuracies in the transition function
- Nevertheless:
  - Markov assumption helps reduce time complexity of algorithms
  - Most, if not all, stochastic processes can be modeled as Markov processes

9

## Reward Function

- **Formally:**

- Define  $R(s, a, s')$  for all  $s, s' \in S$  and for all  $a \in A$ .

Other possible ways to define reward functions

- **Alternate forms:** (given  $R(s, a, s')$  above)
  - $R(s, a)$  as  $\sum_{s'} P(s'|s, a)R(s, a, s')$  independent of  $s'$
  - $R(s)$  as  $\sum_{s'} P(s'|s)R(s, a, s')$ , independent of  $a$  and  $s'$

- **Challenges**

- It is hard to construct reward functions with multiple attributes
- Balance risk vs reward

10

## Solving MDPs

Deriving policies – actions to take at each state

11

## Solving MDPs

- What does a solution look like?

- A policy  $\pi(s): S \rightarrow A$  is a function from states to actions:
  - For every state  $s$ , outputs an appropriate action  $a$ .
- Quality of policy – measured by expected utility of possible state sequence generated by the policy
- Optimal policy  $\pi^*$  is a policy that generates highest expected utility

- An MDP agent:

- Given optimal policy  $\pi^*$ : Agent decides what to do by consulting its current percept, which tells it the current state  $s$ , and then executing action  $a^* = \pi^*(s)$
- The (optimal) policy represents the agent function explicitly – how to behave!

12

### Example: Balancing Risk and Reward

Source: RN Figure 16.2

13

### Finite and Infinite Horizon Problems

- Finite horizon:** Fixed time  $N$  and terminate
  - Return is usually the addition of rewards over the sequence
    - $U_h([s_0, a_0, s_1, a_1, s_2, s_3 \dots s_{N+k}]) = U_h([s_0, a_0, s_1, a_1, s_2, s_3 \dots s_N])$
    - $U_h([s_0, a_0, s_1, a_1, s_2, s_3 \dots s_N]) = R(s_0, a_0, s_1) + R(s_1, a_1, s_2) + \dots + R(s_{N-1}, a_{N-1}, s_N)$
  - Optimal action in a given state can change over time  $N$ , depending on remaining steps
  - Nonstationary** optimal policy:  $\pi_t^*$
- Infinite horizon**
  - No fixed deadline
  - No reason to behave differently in the same state at different times
  - Stationary** optimal policy:  $\pi^*$

14

### Rewards in Infinite Horizon Problems

- Infinite horizon – comparing utilities are difficult**
  - Undiscounted utilities can be infinite
- Additive discounted rewards**
  - $U_h([s_0, a_0, s_1, a_1, s_2, s_3 \dots]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \dots$  where  $\gamma \in [0, 1]$  is the discount factor
  - Discounted rewards with  $\gamma < 1$  and rewards bounded by  $\pm R_{max}$ , utility is always finite

$$U_h([s_0, a_0, s_1, a_1, s_2, \dots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1 - \gamma}$$

Allows preference independence assumption

15

### Preference Independence Assumption

- Assumptions**
  - Each transition  $s_t \xrightarrow{a_t} s_{t+1}$  regarded as an attribute of the history  $[s_0, a_0, s_1, a_1, s_2, s_3 \dots]$
  - Preference independence assumption** – preferences between state sequences are stationary
- Stationary preferences**
  - If two histories  $[s_0, a_0, s_1, a_1, s_2, \dots]$  and  $[s'_0, a'_0, s'_1, a'_1, s'_2, \dots]$  begin with the same transition (i.e.,  $s_0 = s'_0, a_0 = a'_0$ , and  $s_1 = s'_1$ )
  - Then the two histories should be preference-ordered the same way as the histories  $[s_1, a_1, s_2, \dots]$  and  $[s'_1, a'_1, s'_2, \dots]$

16

### How to Achieve Finite Rewards?

- 3 possible solutions to achieve finite rewards**
  - With discounted rewards, utility of infinite sequence is finite, if  $\gamma < 1$ , and rewards bounded by  $\pm R_{max}$ , then  $U_h([s_0, a_0, s_1, a_1, s_2, \dots]) = R_{max}/(1 - \gamma)$
  - Environment has terminal states and if there is a **proper policy**, i.e., agent is guaranteed to get to a terminal state, can even use  $\gamma = 1$  in additive rewards. (See counter examples)
  - Compared in terms of average reward obtained per time step
    - Harder to compute and to analyze**

17

### Utility of State and Optimal Policy

- Main ideas:**
  - Utility of sequence: Sum of the discounted rewards obtained during that sequence
  - Comparing expected utilities obtained when executing policies
  - Utility function of state  $U(s)$**  allows selection of optimal action using MEU
- Assumptions:**
  - Define random variable  $S_t$ : State reached at time  $t$  when executing a particular policy  $\pi$ ;  $S_0 = s$
  - Probability distribution over state sequences  $S_1, S_2, \dots$  determined by: Initial state  $s$ , policy  $\pi$ , and transition model  $T$  for the environment
- Expected utility of executing  $\pi$  starting from  $s$ :**

$$U^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})]$$
  - With  $S_0 = s$ , expectation wrt distribution of state sequences determined by  $\pi$  and  $s$ .

18

## Utility of State and Optimal Policy

- Utility of state (or value of state) is the value of optimal policy

$$U(s) = U^{\pi^*}(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')] = V(s)$$

- Expected sum of discounted rewards if an optimal policy is executed
  - $R(s) = \sum_a P(s'|s, a)R(s, a, s')$  is the "short term" reward for being in  $s$
  - $U(s) = V(s)$  is the "long term" total expected reward from  $s$  onward
- Optimal action selected through maximizing utility of state  $U(s)$  based on MEU:

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')]$$

- Optimal policy is independent of the starting state in infinite horizon problems with discounted utilities

19

## Bellman Equation: Infinite Horizon

- Definition for infinite horizon problem:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')] \text{ OR}$$

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a)U(s')$$

- $U(s) = U^{\pi^*}(s)$ : The utility of a state is the expected reward for the next transition (or the current state) plus the discounted utility of the next state, assuming optimal action taken

- Solution

- The utilities of the states as defined as the expected utility of subsequent state sequences are the unique solutions of the set of Bellman equations
- Gives direct relationship between utility of a state and the utility of its neighbors.
- There is 1 Bellman equation per state. So,  $|S|$  nonlinear equations (due to max) with  $|S|$  unknowns (utility of states).

20

## Q-Function

- Action-Utility Function or Q-Function
  - $Q(s, a)$ : Expected utility of taking a given action in a given state

$$U(s) = \max_a Q(s, a)$$

- Computing optimal policy:
 
$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

- Bellman Equation for Q-functions

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

- Q-Value Function

- Inputs: mdp, s, a, U
- Return:  $\sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')]$

Used in the  
algorithms later

21

21

 NUS Computing

## Markov Decision Process

CS4246/CS5446  
AI Planning and Decision Making

1

## Value Iteration

Solution Method

2

## Value Iteration

- Repeatedly perform the Bellman Update

$$U_{t+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U_t(s')] \quad \text{OR}$$

$$U_{t+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a)U_t(s')$$

- Simultaneous updates of all states:
  - $|S|$  nonlinear equations (due to max) with  $|S|$  unknowns (utility of states).
- Guaranteed to reach an equilibrium through convergence
- Final utility values must be unique solutions to the Bellman equations
- Corresponding policy is optimal

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')]$$

3

## Value Iteration Algorithm

```

function VALUE-ITERATION(mdp, ϵ) returns a utility function
 inputs: mdp, an MDP with states S , actions $A(s)$, transition model $P(s'|s, a)$,
 rewards $R(s, a, s')$, discount γ
 ϵ , the maximum error allowed in the utility of any state
 local variables: U , U' , vectors of utilities for states in S , initially zero
 δ , the maximum relative change in the utility of any state
 repeat
 $U \leftarrow U'$; $\delta \leftarrow 0$
 for each state s in S do
 $U'[s] \leftarrow \max_{a \in A(s)} Q\text{-VALUE}(\text{mdp}, s, a, U) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')]$
 if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$
 until $\delta \leq \epsilon(1-\gamma)/\gamma$
 return U

```

Source: RN Figure 16.6

4

## Some Terminology

- Definitions:
  - A **fixed point** is any input unchanged by a function
  - A **contraction** is a function, when applied to two inputs, produces two outputs that are closer together
    - E.g., "divide by two" is a contraction
    - A **contraction has only one fixed point**
    - When contraction is applied to any argument, the value must get closer to the fixed point; repeated application reaches fixed point in the limit
  - Max norm** or distance between two vectors is the maximum difference between any two corresponding elements
    - It measures "length" of a vector by the absolute value of its biggest component

5

## Convergence

**Bellman update operator**

- Value iteration:
  - Converges to the (unique) utility function for discounted problems with  $\gamma < 1$
  - The Bellman update  $U_{t+1} \leftarrow BU_t$ , is a **contraction** by a factor of  $\gamma$  on the space of utility vectors:

$$\|BU_t - BU'_t\| \leq \gamma \|U_t - U'_t\|$$

- where **max norm**  $\|U\| = \max_s U(s)$ ;  $\|U - U'\| = \max_s \|U(s) - U'(s)\|$
- For Bellman equations, the utility or value function  $U$  is a **fixed point**

$$U = BU$$

6

5

6

## Convergence

- Repeated application of a contraction reaches a unique fixed point  $U$   
 $\|U_{t+1} - U\| = \|BU_t - BU\| \leq \gamma \|U_t - U\| \leq \gamma^t \|U_0 - U\|$  for any initial  $U_0$
- Reaching fixed point:
  - $U = BU$  is the fixed point utility function
  - Error  $\|U_t - U\|$  reduced by a factor of at least  $\gamma$  on each iteration; converges exponentially fast
  - Utilities of all states bounded by  $\pm R_{max}/(1 - \gamma)$ :  
 $\|U_0 - U\| \leq 2R_{max}/(1 - \gamma)$

7

## Convergence

- Factors influencing convergence:
  - $N$  iterations to reach error of at most  $\epsilon$ :  
 $\|U_N - U\| \leq \gamma^N \cdot 2R_{max}/(1 - \gamma) \leq \epsilon$
- Termination condition:
  - If the update is small (i.e., no state utility changes by much), then the error, compared with the true utility function, also is small
  - If  $\|U_{t+1} - U_t\| < \frac{\epsilon(1-\gamma)}{\gamma}$  then  $\|U_{t+1} - U\| < \epsilon$

8

7

8

## Convergence: Calculations

- If we run  $N$  iterations, we get:  

$$\|U_N - U\| \leq \frac{\gamma^N R_{max}}{1-\gamma}$$
- To get error at most  $\epsilon$ , we have:  

$$\frac{\gamma^N R_{max}}{1-\gamma} \leq \epsilon$$
  

$$\gamma^N R_{max} \leq \epsilon(1-\gamma)$$
  

$$\frac{R_{max}}{\epsilon(1-\gamma)} \leq \frac{1}{\gamma^N} = \left(\frac{1}{\gamma}\right)^N$$
- Take log:  

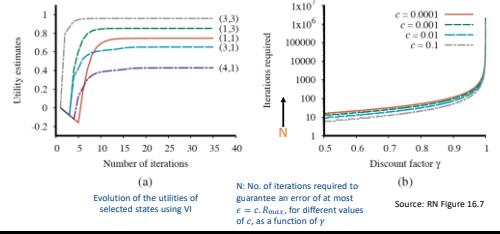
$$N \log\left(\frac{1}{\gamma}\right) \geq \log\left(\frac{R_{max}}{\epsilon(1-\gamma)}\right)$$
  

$$N = \left\lceil \frac{\log\left(\frac{2R_{max}}{\epsilon(1-\gamma)}\right)}{\log\left(\frac{1}{\gamma}\right)} \right\rceil$$
- Terminating condition:
  - $\|U_{t+1} - U_t\| \leq \epsilon(1-\gamma)/\gamma$
  - $\Rightarrow \|U_{t+1} - U\| \leq \epsilon$

9

Remember: Values at each state bounded by  $\pm \frac{R_{max}}{1-\gamma}$ .  
The 2 in the numerator reflects the bounds

## Example: Convergence



Source: RN Figure 16.7

10

9

10

## Computing Optimal Policy

Would an MEU policy based on the estimated utilities behave optimally?

- Policy Loss
  - Let  $\pi_i$  be the MEU policy wrt  $U_i$
  - Recall:  $U^{\pi_i}(s)$  is utility obtained if  $\pi_i$  is executed starting in state  $s$
  - Policy loss  $\|U^{\pi_i} - U\|$  is the max loss by following  $\pi_i$  instead of  $\pi^*$ .
- Connecting utility error and policy loss
  - If  $\|U_i - U\| < \epsilon$  then  $\|U^{\pi_i} - U\| < 2\epsilon$
- In practice:
  - $\pi_i$  often becomes optimal long before convergence of  $U_i$

11

11

## Policy Iteration

Solution Method

12

12

## Motivation for Policy Iteration

- Value iteration:**
  - Exact algorithm
  - Scales poorly
- Observations:**
  - If one action is clearly better than all others, then exact magnitude of utilities on the states need not be precise
  - Utility function estimate can be inaccurate to derive optimal policy
  - There is another way to find optimal policies: **Policy Iteration**

13

## Policy Iteration

- Begin with Initial policy:  $\pi_0$**
- Alternate between:**
  - Policy evaluation:**
    - Given a policy  $\pi_i$ , calculate  $U_i = U^{\pi_i}$ , utility of each state if  $\pi_i$  is executed
    - $U_i(s) = \sum_{s'} P(s'|s, \pi_i(s)) [R(s, \pi_i(s), s') + \gamma U_i(s')]$
  - Policy improvement:**
    - Calculate new MEU policy  $\pi_{i+1}$  using one-step look-ahead based on  $U_i$ .
- Terminate when there is no change in utilities for policy improvement step**
  - Reaching fixed point of the Bellman update
  - Finite number of policies, hence algorithm must terminate
- Complexity:**
  - Assuming  $|S| = n$  linear equations with  $n$  unknowns—can be solved in  $O(n^2)$  time.

14

13

14

## Policy Iteration Algorithm

```

function POLICY-ITERATION(mdp) returns a policy
 inputs: mdp, an MDP with states S, actions A(s), transition model $P(s' | s, a)$
 local variables: U, a vector of utilities for states in S, initially zero
 π , a policy vector indexed by state, initially random

 repeat
 U \leftarrow POLICY-EVALUATION(π, U, mdp)
 unchanged? \leftarrow true
 for each state s in S do
 $a^* \leftarrow \text{argmax } Q\text{-VALUE}(mdp, s, a, U) = \sum_{a'} P(s'|s, a)[R(s, a, s') + \gamma U(s')]$
 if $Q\text{-VALUE}(mdp, s, a^*, U) > Q\text{-VALUE}(mdp, s, \pi[s], U)$ then
 $\pi[s] \leftarrow a^*$; unchanged? \leftarrow false
 until unchanged?
 return π

```

Source: RN Figure 17.9

15

15

## Example: Implementing Policy Evaluation

- Policy evaluation**

$$U_i(s) = \sum_{s'} P(s'|s, \pi_i(s)) [R(s, \pi_i(s), s) + \gamma U_i(s')] \\ \text{No max operator} \quad = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

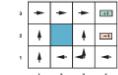
$$U_i(1,1) = 0.8[-0.04 + U_i(1,2)] + 0.1[-0.04 + U_i(2,1)] + 0.1[-0.04 + U_i(1,1)] \\ U_i(1,2) = 0.8[-0.04 + U_i(1,3)] + 0.2[-0.04 + U_i(1,2)]$$

- Policy evaluation equations are linear equations**

- Simplified Bellman equation
- For  $n$  states, can be solved in  $O(n^2)$  time
- For large state-spaces, use iterative method:

$$U_{t+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, a) U_t(s')$$

Source: RN 16.2.2



16

16

## Policy Iteration: Why Does It Work?

- Termination**
  - Policy improvement step yields no change in utilities
  - Utility function  $U_i$  is a fixed point of Bellman update, and a solution to the Bellman equations
  - $\pi_i$  must be an optimal policy
  - Only finitely many policies for a finite state space, each iteration can be shown to yield a better policy, hence policy iteration must terminate
- Correctness**
  - Follows from Policy Improvement Theorem [SB 4.2]

17

17

## Policy Improvement Theorem

- Let:**

- $Q^\pi(s, a) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U^\pi(s')]$  be the action-utility function of  $\pi$  (one-step look-ahead using action  $a$ )

- Theorem:** [Proof in SB 4.2]

- Let  $\pi$  and  $\pi'$  be any pair of deterministic policies such that for all  $s \in S$ ,

$$Q^\pi(s, \pi'(s)) \geq U^\pi(s) \quad \text{Recall: } U(s) = \max_a Q(s, a)$$

- Then

$$U^{\pi'}(s) \geq U^\pi(s) \text{ for all } s \in S$$

- If the inequality of  $Q^\pi(s, \pi'(s)) \geq U^\pi(s)$  is strict for any state  $s$ , then corresponding inequality for  $U^{\pi'}(s) \geq U^\pi(s)$  is strict for that  $s$

Source: Sutton and Barto, 2018/2020 Section 4.2

18

18

## Policy Iteration

- **Modified policy iteration:**

- Do only  $k$  iterations of (simplified) value iteration instead of reaching convergence
- Approximate evaluation

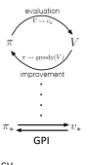
- **Asynchronous policy iteration:**

- Pick only a subset of states for policy improvement or for updating in policy evaluation
- Converges as long as continuously update all states

- **Generalized policy iteration:** (SB) 4.6

- Update utility according to policy, and improve policy wrt the utility function.
- VI, PI, asynchronous policy iteration are all special cases.

- May interleave in different ways based on conditions



19

## Markov Decision Processes

### Online Algorithms

CS4246/CS5446  
AI Planning and Decision Making

1

## Curse of dimensionality

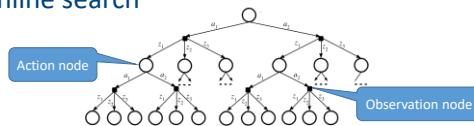
- State space grows exponential with the number of variables

- VI/PI iterates through all states; so exponential with the number of variables
- To handle curse of dimensionality
  - Use function approximation
    - Linear function of features
    - Deep neural networks
  - Do online search with sampling

© Copyright National University of Singapore. All Rights Reserved.
2

2

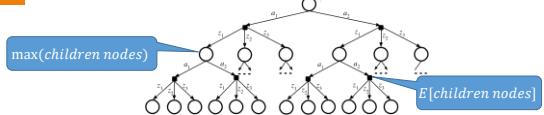
## Online search



- At every step, construct a search tree
  - Up to a fixed depth  $D$
  - Root is the current state
  - $|A|$  children of the root (and other action nodes)
  - $|S|$  children of observation nodes

© Copyright National University of Singapore. All Rights Reserved.
3

## Online search



- To compute the value at the root:
  - Initialize leaf with value estimates (or zeros)
  - At observation node, compute the expected values of the children
  - At the action nodes, compute the max of the children

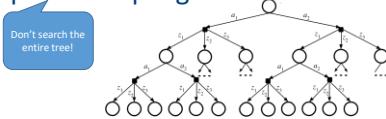
Q: Have we fixed the curse of dimensionality?

© Copyright National University of Singapore. All Rights Reserved.
4

4

3

## Sparse sampling



- Tree size:  $|A|^D |S|^D$
- Sparse sampling<sup>1</sup>
  - Estimate by sampling  $k$  observations at observation nodes instead of using  $|S|$  states as possible observations
    - Tree size now:  $|A|^D k^D$
    - Curse of dimensionality is solved ...
      - ... but still exponential with the search depth – curse of history

© Copyright National University of Singapore. All Rights Reserved.
<sup>1</sup>Michael Kearns, Yishay Mansour, and Andrew Y Ng, "A sparse sampling algorithm for near-optimal planning in large Markov decision processes", In: Machine learning 49(2-3) (2002), pp. 193-208.
5

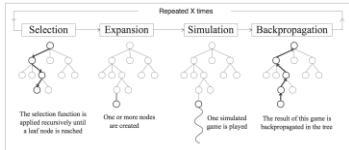
## Monte Carlo Tree Search

© Copyright National University of Singapore. All Rights Reserved.
6

5

6

## MCTS<sup>2</sup>



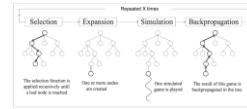
- Commonly used to solve MDP and games

© Copyright National University of Singapore. All Rights Reserved.

Guillaume Chaillet et al. "Monte-Carlo Tree Search: A New Framework for Game AI", In: AIDE, 2008.

7

## MCTS



- Repeatedly run trials from the root (current state in online search)

- Trial:

- Repeatedly select the node to go to at the next level until
  - Target depth is reached, or
  - Selected node has not been discovered – create a new node; run a simulation using a default policy till required depth
- Backup the outcomes all the way to the root

- Anytime policy: When time is up, use the action that looks best at the root at that time

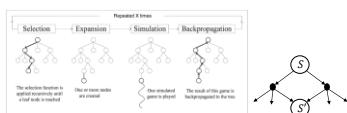
© Copyright National University of Singapore. All Rights Reserved.

8

7

8

## MCTS

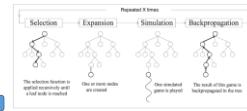


- For MDP, a tree (actually DAG) node  $n$  is associated with a state  $s$
- A node  $n'$  at the next level is selected by applying an action  $a$  to  $s$ 
  - Sample next state  $s'$  (corresponding to  $n'$ ) according to  $P(s'|s, a)$
- Action  $a$  is selected to balance exploration with exploitation

© Copyright National University of Singapore. All Rights Reserved.

9

## MCTS



- Estimated value  $\hat{V}(n)$  at node  $n$  is the average return of all trials at  $n$ 
  - Return  $r_t(n)$  of trial  $t$  starting from  $n$  with state  $s$  and next node  $n'$
  - $\hat{r}_t(n) = R + \gamma r_{t+1}(n')$
- Estimated  $Q$ -function at  $n$ ,  $\hat{Q}(n, a)$ , is the average return of all trials at  $n$  that start with action  $a$ 
  - $\hat{Q}(r, a)$  at the root  $r$  is used to select the action to take at the root node
- All these are updated in the backup operation to the root

© Copyright National University of Singapore. All Rights Reserved.

10

9

10

## Upper confidence tree (UCT)

## UCT<sup>3</sup>

- Action selection is guided by this function:
 
$$\pi_{UCT}(n) = \arg \max_a \hat{Q}(n, a) + c \sqrt{\frac{\ln(N(n))}{N(n, a)}}$$
  - Being greedy
- Where,  $N(n)$  and  $N(n, a)$  count the number of trials through  $n$  and  $(n, a)$  respectively and  $c$  is a constant
- UCT will eventually converge to the optimal policy with enough trials  $D - 1$  times
  - Worst case can be very bad<sup>4</sup>  $\Omega(\exp(\exp(\dots \exp(1) \dots)))$
  - Often works well in practice
    - E.g., PROST planner<sup>5</sup>, which uses UCT, won the international probabilistic planning competition in 2011 & 2014

© Copyright National University of Singapore. All Rights Reserved.

11

© Copyright National University of Singapore. All Rights Reserved.

12

12

<sup>3</sup>Gábor Kocsis and Csaba Szepesvári. "Bandit based monte-carlo planning". In: European conference on machine learning. Springer, 2006, pp. 282-293.

## MCTS in practice

Player 2 uses MCTS

- Visualizing MCTS:

- [https://www.youtube.com/watch?v=vFvRSxNLTg7U&ab\\_channel=DaveDyer](https://www.youtube.com/watch?v=vFvRSxNLTg7U&ab_channel=DaveDyer)

13

## AlphaGo Zero – MCTS

- Variant of UCT that exploits the policy head output:  $P(s, a)$

$$\pi_{UCT}(s) = \arg \max_a \hat{Q}(s, a) + c P(s, a) \sqrt{\frac{\sum_b N(s, b)}{1 + N(s, a)}}$$

- When leaf node is reached, the value head output is used to evaluate the state instead of doing a roll-out (simulation)

- Go is a zero-sum, turn taking game instead of MDP

- Search alternates between:
  - selecting action that maximizes when it is first player's turn
  - selecting the action that minimizes for second player's turn

- At termination: reward +1 for the first player win and -1 for second player win

© Copyright National University of Singapore. All Rights Reserved.

14

14

## AlphaGo Zero – Approximate PI

- Recollect: Policy iteration has 2 stages, policy evaluation and policy improvement
- AlphaGo Zero does both using supervised learning
- With the current value function, MCTS can be viewed as a policy improvement operator – gives improved policy values for the evaluated states
- Self-play with search gives the policy evaluation for the evaluated states
- Supervised learning is used to interpolate the values and the policy over the whole domain using data from a set of states

© Copyright National University of Singapore. All Rights Reserved.

15

15

 NUS Computing

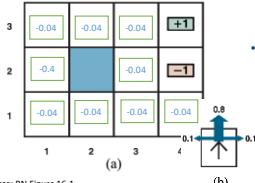
## Reinforcement Learning

CS4246/CS5446  
AI Planning and Decision Making

1

### Example: Navigation in Grid World

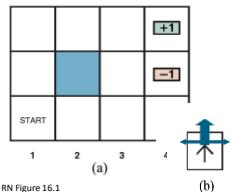
- Transition model:
  - Define (stochastic) outcome of each action
  - $P(c|s, a)$  – probability of reaching state  $s'$  if action  $a$
- Reward model:
  - Define reward received for every transition
  - $R(s, a, s')$  – For every transition from  $s$  to  $s'$  via action  $a$ ; AND/OR
  - $R(s)$  – For any transition into state  $s$
  - Rewards may be positive or negative, but bound by  $-1 \leq R \leq +1$
  - Utility function  $U(s)$  depends on the sequence of states and actions – environment history – sum of rewards of the states in the sequence



Source: RN Figure 16.1

2

### Example: Navigation



Source: RN Figure 16.1

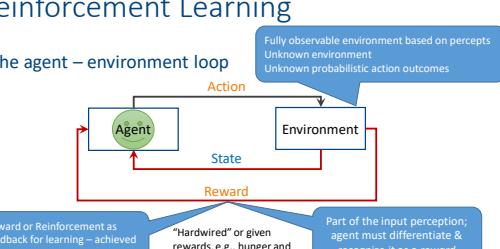
3

### Reinforcement Learning

- Decision making in complex and uncertain environments
  - Learning to behave proficiently in unfamiliar environment, given only percepts and occasional rewards
  - E.g., playing chess, flying helicopter, exploring Mars, assisting elderly
- Assume:
  - Environment is a Markov Decision Process (MDP)
  - Optimal policy – policy that maximizes the expected total reward
- Reinforcement learning
  - Use observed rewards to learn an optimal policy for the environment
  - Often with no knowledge of the environment model or reward function
    - E.g., don't know the game rules, just play until Win/Loss declared
    - Would this for a good approach for an AI tutor?
  - Influences from psychology, neuroscience, operations research, and optimal control theory.

4

### Reinforcement Learning

- The agent – environment loop
 

5

### Types of RL Agents

- Reinforcement learning
  - [PL] Passive Learning: Learning Utility functions
  - [AL] Active Learning: Learning to Plan or Decide
  - [MB] Model-based: Learn transition model to solve problem
  - [MF] Model-free: Do not (need to) learn transition model to solve problem
- Passive learning agent
  - Has a fixed policy that determines his behavior
  - sf **policy evaluation** in policy iteration
- Active learning agent
  - Must decide what actions to take
  - sf **value iteration**

6

## Passive Learning

Predict utility function (value function) of state given a fixed policy

7

## Passive Learning

Main idea:

- Agent executes a set of trials in environment using policy  $\pi$ ; starts in (1,1) until reaches terminal state
- Percepts supply both current state and reward received in state
- Use reward information to learn expected utility  $U^\pi(s)$  associated with each non-terminal state  $s$

$$U^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t)]$$

where  $R(s)$  is reward for a state  $s$ ,  $S_t$  is a random variable indicating state reached at time  $t$  when executing policy  $\pi$

- sf policy evaluation (in policy iteration)
- Assume  $\gamma = 1$  in examples that follow

8

## Learning Utility Function

The diagram shows a 4x4 grid world with states labeled 1 to 16. Arrows indicate transitions between states. A utility matrix is shown below:

|        |        |        |        |
|--------|--------|--------|--------|
|        |        |        |        |
| 0.8916 | 0.8078 | 0.8078 | 0.1    |
| 0.8016 |        | 0.7003 | 0.1    |
| 0.7453 | 0.6953 | 0.6514 | 0.4279 |

Source: RN Figure 23.1

- The policy  $\pi$  is fixed:
  - Agent always executes  $\pi(s)$  in state  $s$
- Goal: Learn utility function or value function  $U^\pi(s)$  from observations
  - Unknown transition model,  $P(s'|s, a)$
  - Unknown reward function,  $R(s)$

If known, simply do policy evaluation!

9

## Learning Utility Function

The diagram shows a 4x4 grid world with a 'Start state' indicated by a blue arrow. Three trials are shown starting from (1,1):

- Trial 1: (1,1)  $\xrightarrow{Up}$  (1,2)  $\xrightarrow{Up}$  (1,3)  $\xrightarrow{Right}$  (2,3)  $\xrightarrow{Right}$  (3,3)  $\xrightarrow{Right}$  (4,3)
- Trial 2: (1,1)  $\xrightarrow{Up}$  (1,2)  $\xrightarrow{Up}$  (1,3)  $\xrightarrow{Right}$  (2,3)  $\xrightarrow{Right}$  (3,3)  $\xrightarrow{Right}$  (4,3)
- Trial 3: (1,1)  $\xrightarrow{Up}$  (1,2)  $\xrightarrow{Up}$  (1,3)  $\xrightarrow{Right}$  (2,3)  $\xrightarrow{Right}$  (3,3)  $\xrightarrow{Right}$  (4,2)

Utility or value of a state:

- Expected total reward from that state onwards
- Also called **expected reward-to-go / expected return**

Utility/value of  $s$  under  $\pi$  is:

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t) | S_0 = s\right]$$

Assume  $\gamma = 1$

10

## Monte Carlo Learning

The diagram shows a 4x4 grid world with a path highlighted in red. Visit counts are shown in circles:

|       |     |       |       |       |       |       |     |       |       |       |       |       |       |       |
|-------|-----|-------|-------|-------|-------|-------|-----|-------|-------|-------|-------|-------|-------|-------|
| (1,1) | .04 | (1,2) | .04   | (1,3) | .04   | (1,2) | .04 | (1,3) | .04   | (2,3) | .04   | (3,3) | +1    | (4,3) |
| Up    | Up  | Up    | Right | Right | Right | Up    | Up  | Up    | Right | Right | Right | Right | Right |       |
| (1,1) | .04 | (1,2) | .04   | (1,3) | .04   | (2,3) | .04 | (3,3) | .04   | (3,2) | .04   | (3,3) | +1    | (4,3) |
| Up    | Up  | Up    | Right | Right | Right | Up    | Up  | Up    | Right | Right | Right | Right | Right |       |
| (1,1) | .04 | (1,2) | .04   | (1,3) | .04   | (2,3) | .04 | (3,3) | .04   | (3,2) | .04   | (3,3) | -1    | (4,2) |
| Up    | Up  | Up    | Right | Right | Right | Up    | Up  | Up    | Right | Right | Right | Right | Up    |       |

- Main idea:
  - Maintain a running average of expected return for each state in a table
  - Run infinitely many trials; value converges to the true expected value
  - Number of visits may be treated differently: e.g., first visit vs. every visit
    - E.g., consider only 0.8 for (1,2); 0.88 is ignored
    - E.g., consider both 0.8 and 0.88 for (1,2)
    - Both converge to the true expected value in the limit

11

## Monte Carlo Learning

Also called **Direct Utility Estimation**:

An instance of supervised learning - Each example has state as input and observed return (unbiased estimate) as output

$$((x_1, y_1), u_1), ((x_2, y_2), u_2), \dots, ((x_n, y_n), u_n)$$

where  $u_j$  is the measured utility of the  $j^{\text{th}}$  example

Very slow convergence:

Need to wait till the end of the episode before learning can begin  
By ignoring constraints, searches much larger space, including utility functions that violate the Bellman equations for a fixed policy:

$$U^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

12

### Example: Adaptive Dynamic Programming

(1,1)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (2,3)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (4,3)  $\begin{matrix} +1 \\ +1 \end{matrix}$   
(1,1)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (2,3)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (3,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (4,3)  $\begin{matrix} +1 \\ +1 \end{matrix}$   
(1,1)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (2,3)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (3,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (4,2)  $\begin{matrix} -1 \\ -1 \end{matrix}$

What is the transition probability function from  $s = (3, 3)$ ?

To compute: Why are all the actions *Right*?

- $P(s' = (3, 2) | s = (3, 3), a = \text{Right}) = \frac{1}{3}$
- $P(s' = (4, 3) | s = (3, 3), a = \text{Right}) = \frac{2}{3}$

13

### Adaptive Dynamic Programming (ADP)

- Learn transition function  $T$  empirically through experience
  - Count action outcomes for each  $s, a$
  - Normalize to give estimates of transition probabilities  $P(s'|s, a)$
- Learn reward function  $R(s, a, s')$  upon entering state  $s'$
- Solve MDP with learned  $T$ 
  - Given  $\pi$ , perform **policy evaluation** for all  $s$ :  

$$U^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$
  - Solve  $|S|$  linear equations with  $|S|$  unknowns in  $O(|S|^3)$  time
- Notes:**
  - Can learn  $P(s'|s, a)$  using supervised learning
  - Exploits constraints among utilities of states

14

### ADP-Learner

```

function PASSIVE-ADP-LEARNER(percept) returns an action
 inputs: percept, a percept indicating the current state s' and reward signal r
 persistent: π , a fixed policy
 mdp, an MDP with model P , rewards R , actions A , discount γ
 U , a table of utilities for states, initially empty
 $N_{s,a,s'}$, a table of outcome count vectors indexed by state and action, initially zero
 s_a , the previous state and action, initially null

 Reward function R
 if s' is new then $U[s'] \leftarrow 0$
 if s is not null then
 increment $N_{s,a,s'}$
 $R[s, a, s'] \leftarrow r$
 add n to $A[s]$

 Transition function T
 $P[s \mid s, a] \leftarrow \text{NORMALIZE}(N_{s,a,s'})$ ← MLE
 $s, a \leftarrow s', \pi[s']$
 return a

```

Source: RN Figure 23.2.

15

### Example: Temporal-Difference (TD) Learning

(1,1)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (2,3)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (4,3)  $\begin{matrix} +1 \\ +1 \end{matrix}$   
(1,1)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (2,3)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (3,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (4,3)  $\begin{matrix} +1 \\ +1 \end{matrix}$   
(1,1)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,2)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (1,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (2,3)  $\begin{matrix} \text{Up} \\ \text{Up} \end{matrix}$  (3,3)  $\begin{matrix} \text{Right} \\ \text{Right} \end{matrix}$  (3,2)  $\begin{matrix} -1 \\ -1 \end{matrix}$  (4,2)

After first trial:  

$$U^\pi(1,3) = \frac{0.84+0.92}{2} = 0.88$$

$$U^\pi(2,3) = 0.96$$

In the long run, must obey:  

$$U^\pi(1,3) = -0.04 + U^\pi(2,3)$$

Hence:  

$$U^\pi(1,3) = 0.92$$
 Current value of 0.88 needs updating!

16

### Temporal Difference Learning (TD)

- Temporal difference learning (TD)
  - For a transition from state  $s$  to  $s'$ , TD learning does:  

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s, \pi(s), s') + \gamma U^\pi(s') - U^\pi(s))$$
  - where,  $\alpha$  is the learning rate
- In TD learning:
  - TD target:  $R(s, \pi(s), s') + \gamma U^\pi(s')$
  - TD term or error:  $R(s, \pi(s), s') + \gamma U^\pi(s') - U^\pi(s)$
  - TD term is error signal, update is intended to reduce error
  - Increases  $U^\pi(s)$  if  $R(s, \pi(s), s') + \gamma U^\pi(s')$  is larger than  $U^\pi(s)$ ; decreases otherwise
  - Converges if  $\alpha$  decreases appropriately with the number of times the state has visited.  
E.g.,  $\alpha(n) = O\left(\frac{1}{n}\right)$

17

### Temporal-Difference (TD) Learning

- Main idea:
  - Adjust utility estimates towards ideal (local) equilibrium with correct utility estimates
  - In passive learning equilibrium is given by the Bellman equation for a fixed policy:  

$$U_i(s) = \sum_{s'} P(s'|s, \pi_i(s)) [R(s, \pi_i(s), s') + \gamma U_i(s')]$$
- TD update:
  - Involves only observed successor  $s'$ , average value of  $U^\pi(s')$  will converge to correct value
  - If learning rate  $\alpha$  is changed from fixed parameter to decreasing function of increasing number of visits to a state, then  $U^\pi(s)$  converges
- TD vs MC and ADP:
  - Exploits more of the Bellman equation constraints than MC
  - Does not need to learn the model (here refers to transition function  $T$ ), unlike ADP

18

## TD-Learner (Model-Free RL)

```

function PASSIVE-TD-LEARNER(percept) returns an action
 inputs: percept, a percept indicating the current state s' and reward signal r
 persistent: π , a fixed policy
 s, the previous state, initially null
 U, a table of utilities for states, initially empty
 Ns, a table of frequencies for states, initially zero

 if s' is new then U[s'] \leftarrow 0
 if s is not null then
 increment Ns[s]
 U[s] \leftarrow U[s] + $\alpha(N_s[s]) \times (r + \gamma U[s'] - U[s])$
 s \leftarrow s'
 return $\pi[s']$

 • TD needs only observed transitions to do the updates; no transition model is needed!
 • Learn from every experience – update each time we experience s, $\pi(s)$, and s'

```

Source: RN Figure 23.4

19

## ADP vs TD/MC

- ADP**
  - Learns the model and then solves for the value
- TD/MC**
  - Don't need a model
  - Can work with measurements from real-world or simulator
- ADP**
  - More data efficient
  - Requires less data from the real-world
- TD**
  - Doesn't need to compute expectation
  - Computationally more efficient

20

## Active Learning

Choose to act optimally based on prediction of utility or value of states  
Exploration vs exploitation, Q-learning, SARSA

21

## Active Learning

- Main ideas:**
  - T* and *R* unknown; can choose any action *a* at each step
  - Goal: Learn optimal policy  $\pi^*$  that obey the Bellman equations:
$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')]$$
- In summary:**
  - Learner makes choices
  - Tradeoff between exploration vs. exploitation:
  - If learned  $\hat{T}$  is no the true *T*, optimal policy obtained using  $\hat{T}$  may be suboptimal wrt *T*, possibly incurring large **policy loss**

22

## From Passive to Active ADP-Learner

```

function PASSIVE-ADP-LEARNER(percept) returns an action
 inputs: percept, a percept indicating the current state s' and reward signal r
 persistent: π , a fixed policy
 mdp, an MDP with model P, rewards R, actions A, discount γ
 U, a table of utilities for states, initially empty
 Ns,a,s', a table of outcome count vectors indexed by state and action, initially zero
 s, a, the previous state and action, initially null

 Reward function R
 Transition function T

 If s' is new then U[s'] \leftarrow 0
 If s is not null then
 increment Ns,a,s'[s, a][s']
 R[s, a, s'] $\leftarrow r$
 P[s, a, s'] \leftarrow T[s, a, s']
 P[s, a, s'] \leftarrow NORMALIZE(Ns,a,s'[s, a])
 U \leftarrow POLICY-EVALUATION(π , U, mdp)
 s, a \leftarrow s', π[s']
 return a

 • Replace POLICY-EVALUATION step by $\pi \leftarrow \text{POLICY-ITERATION}(\text{mdp})$ above
 • π is no longer fixed; changes as transitions & rewards are learned

```

Source: RN Figure 23.2

23

## Active Adaptive Dynamic Programming

- For active-ADP:**
  - Learn model *T* with outcome probabilities for all actions, not just for a fixed policy.
  - Learn utilities defined by optimal policy that obey the Bellman equation
  - Then, run VI or PI to determine optimal policy
  - Extract an optimal action by one-step look-ahead to maximize expected utility
- Note:**
  - Learner has choice of actions
  - Actions don't just maximize rewards for given current learned model *T*, they also affect the percepts (states and rewards) received
  - Should learner then just execute action recommended by optimal policy? Why?

24

## Greedy ADP Learner

Source: RN Figure 23.6

- Resulting algorithm is greedy w.r.t. the policy
- Policy resulting from executing action suggested by the learned model
  - Note, for e.g., at (2, 1) it goes right instead of left
  - Policy reasonable, but not optimal
- Agent is greedy – sometimes doesn't try better policies sufficiently after finding a suboptimal one

25

## Exploration Vs Exploitation

Hope to learn something new about the problem!

- Actions in RL gain reward and help to learn a better model
  - By improving the model, better reward may potentially be obtained
- Need to trade off
  - Exploration:** Choose action to learn the true model  $T$  (by affecting the states and rewards received) to receive greater rewards in future
  - Exploitation:** Given current learned model  $T$ , choose action to maximize the reward (as reflected in current utility estimates)
- Main questions:
  - How to balance exploration and exploitation to maximize long-term expected rewards?
  - When to “stop” learning the model?
- With greater understanding, less exploration is necessary!

26

## Greedy in the Limit of Infinite Exploration

- GLIE: A scheme for balancing exploration and exploitation
  - Try each action in each state an unbounded number of times to avoid a finite probability of missing an optimal action
  - Eventually become greedy so that it is optimal w.r.t. the true model
- GLIE-based schemes for forcing exploration in RL:
  - $\epsilon$ -greedy exploration
    - Choose the greedy action with probability  $(1 - \epsilon)$
    - Choose a random action with probability  $\epsilon$
  - GLIE based  $\epsilon$ -greedy eventually converges to the optimal, but can be slow
    - One solution: lower  $\epsilon$  over time (e.g.,  $\epsilon = \frac{1}{t}$ )
    - Another solution: use exploration functions!

27

## Optimism

- Alternatively, balance exploration and exploitation using greedy action selection w.r.t. an optimistic estimate of the utility,  $U^+(s)$
- One way to construct  $U^+(s)$  is to use an exploration function  $f(u, n)$  with the following update:
 
$$U^+(s) \leftarrow \max_a f\left(\sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U^+(s)], N(s, a)\right)$$
  - Where,  $N(s, a)$  is the number of times action  $a$  has been tried in state  $s$

28

## Exploration Functions

- When to explore:
  - Random actions: explore a fixed amount
  - Better idea: explore areas where “badness” has not been established
- Exploration function:  $f$ 
  - Takes a value estimate ( $u$ ) and a count ( $n$ ), and returns an optimistic utility  $f(u, n)$
  - Determines how greed (preference for high values of  $u$ ) is traded off against curiosity (preference for actions that have not been tried often with low  $n$ )
  - $f$  should be increasing in  $u$  and decreasing in  $n$  (form is not important)

29

## Example: Exploration Functions

- An exploration function  $f(u, n)$  that is increasing with  $u$  and decreasing with  $n$ :

$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

Where:

- $R^+$  is an optimistic estimate of the best possible reward
- $N_e$  is the parameter

Agent tries each state-action pairs  $N_e$  times

- Example: **RMAX** [1]
  - Works well for small problems, has guarantees

[1] Bratman, R.J. and M. Tennenholtz, *R-max - a general polynomial time algorithm for near-optimal reinforcement learning*. J. Mach. Learn. Res., 2003. 3(null): p. 213-231.

30

## Q-Learning: Learning Action-Utility Functions

- Q-function – Action-utility function  $Q(s, a)$** 
  - Expected total discounted reward if action  $a$  is taken in state  $s$
  - $Q$ -functions (also called **Q-values**) are related to utilities by:  $U(s) = \max_a Q(s, a)$
- Q-Learning**
  - Model-free** learning: Agent can act optimally knowing  $Q$ -function; no need for look-ahead based on a transition model
  - The Bellman equation for  $Q$ -functions:  

$$Q(s, a) = \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma \max_{a'} Q(s', a')]$$
  - Uses TD update to learn action-utility functions or Q-functions:  

$$Q(s, a) \leftarrow Q(s, a) + \alpha [R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

31

31

## Q-learning: Active TD Learning

```

function Q-LEARNING-AGENT(percept) returns an action
 inputs: percept, a percept indicating the current state s' and reward signal r
 persistent: Q , a table of action values indexed by state and action, initially zero
 N_{sa} , a table of frequencies for state-action pairs, initially zero
 s, a , the previous state and action, initially null

 TD Update
 if s is not null then
 increment $N_{sa}[s, a]$
 $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
 $s, a \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$
 return a

 • TD Q-learning does not need a transition model, $P(s'|s, a)$ for learning or action selection

```

Source: RN Figure 23.8  
32

32

## SARSA

- State-Action-Reward-State-Action (SARSA):**
  - Uses TD for prediction (evaluation) and  $\epsilon$ -greedy for action selection  

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a, s') + \gamma Q(s', a') - Q(s, a)]$$
 where  $a'$  is the action taken at state  $s'$
- Note:**
  - Rule is applied at the end of each quintuple  $(s, a, r, s', a')$
  - With appropriately decreasing  $\epsilon$ , SARSA converges to the optimal policy

33

33

## Q-Learning vs SARSA

```

1: controller Q-learning(S, A, γ, α)
2: Input: S is a set of states
3: A is a set of actions
4: γ is the discount factor
5: α is the step size
6: Local
7: # array $Q[S, A]$
8: # states s, s'
9: # action a, a'
10: initialize $Q[S, A]$ arbitrarily
11: observe current state s
12: repeat
13: choose an action a
14: do (a)
15: observe reward r and state s'
16: $Q[s, a] := Q[s, a] + \alpha^\gamma (r + \gamma * \max_{a'} Q[a', a'] - Q[s, a])$
17: $s := s'$
18: until termination
19:

```

Figure 12.3: Q-learning controller

```

1: controller SARSA(S, A, γ, α)
2: Input: S is a set of states
3: A is a set of actions
4: γ is the discount factor
5: α is the step size
6: Local
7: # array $Q[S, A]$
8: # state s, s'
9: # action a, a'
10: initialize $Q[S, A]$ arbitrarily
11: observe current state s
12: select an action a using a policy based on Q
13: reward r
14: do (a)
15: observe reward r and state s'
16: select an action a' using a policy based on Q
17: $Q[s, a] := Q[s, a] + \alpha^\gamma (r + \gamma * Q[s', a'] - Q[s, a])$
18: $s := s'$
19: $a := a'$
20: until termination
21:

```

Figure 12.5: SARSA: on-policy reinforcement learning

Source: Poole and Macmark, AI Foundations of Computational Agents, 2e, 2018  
34

34

## On-policy vs Off-policy Learning<sup>1</sup>

- On-policy learning** – data used for learning is generated by the policy being learned
- Situations where data isn't generated by the policy currently being learned**
  - Data collected using currently running policy
  - Explore using a different policy
- In off-policy methods:**
  - Generate data using a behavior policy
  - Try to learn a target policy

<sup>1</sup>Material for this section are mostly from Sutton & Barto  
35

35

## Off-Policy and On-Policy Learning

- Q-learning update rule:**  

$$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a))$$
- SARSA update rule:**  

$$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s, a, s') + \gamma Q(s', a') - Q(s, a))$$
- Q-learning is off-policy** – Target of Q-learning uses the max over possible actions  $a'$  at  $s'$ 
  - Doesn't need the actual action in the next state in the update
  - Can converge to the optimal policy even when trained off-policy
    - Needs appropriately decaying  $\alpha$
    - Each action is taken in each state infinitely often
- SARSA is on-policy** – Target uses  $Q$ -function from the policy that is running
  - Using  $Q$ -function from another state-action pair will give incorrect target value

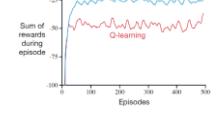
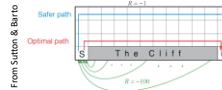
36

## Q-Learning vs SARSA

- **Similarities:**
  - If agent is greedy and always takes action with the best  $Q$ -function: the two are identical
  - Both converge slowly as local updates do not enforce consistency among all the  $Q$ -functions via model
- **Differences:**
  - $Q$ -learning backs up  $Q$ -function from the best action in  $s'$
  - SARSA waits until an action is taken and backs up  $Q$ -function for that action
  - If exploration yields a negative reward: SARSA penalizes the action,  $Q$ -learning does not

37

## Example: Cliff Walking



Source: SB: Example 6.6

- Actions: left, right, up, down
- Reward: -1 for each step; -100 if you fall off
- SARSA &  $Q$ -learning used with  $\epsilon$ -greedy;  $\epsilon = 0.1$ 
  - Why does SARSA outperform  $Q$ -learning?
  - What happens if GLIE exploration is used?

38

 NUS Computing

## “Real World” Reinforcement Learning

CS4246/CS5446  
AI Planning and Decision Making

1

### Scaling

- Number of states grow exponentially with no. of state variables (features)
- Tabular representation (for utility function and  $Q$ -function) scales to tens of thousands of states
  - e.g., Number of states in Backgammon & Chess are of the order of  $10^{20}$  &  $10^{40}$
  - Still considered relatively small as compared to real-world problems!
  - Cannot visit all the states infinitely often
- Approaches to scaling up:
  - Function approximation – approximating utility (value) functions
  - Policy search – systematic search for good policies

2

### Function Approximation

Linear:  
 Approximate Monte Carlo Learning  
 Approximate temporal difference (TD) learning

Non-linear:  
 Deep neural networks

3

### Function Approximation

- Function approximation constructs compact representation of true utility (value) function and  $Q$ -function
  - Example: Represent evaluation function for chess as a linear function of features (or basis functions)  
 $\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \dots + \theta_n f_n(s)$
  - Function approximation uses the  $n$  no. of  $\theta$  parameters to represent a function over a very large number of states
  - RL agent learns  $\theta$  that best approximate the evaluation (utility) function

4

### Function Approximation

- Function approximation allows generalization from small number of states observed in training data to entire state space
  - Example: Backgammon agent learned to play as well as the best human players by observing only  $\approx 10^{12}$  states out of  $10^{20}$  states
- Caveat:
  - If  $n$  (no. of parameters) is too small, may fail to achieve good approximation

5

### Linear Function Approximation

Approximate Monte Carlo Learning  
 Approximate temporal difference (TD) learning

6

## Example: Navigation in Grid World

- Use:  $\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$
- If  $(\theta_0, \theta_1, \theta_2) = (0.5, 0.2, 0.1)$ , then  $\hat{U}(1,1) = 0.8$
- Given a collection of trials:
  - Obtain a set of sample values of  $\hat{U}_\theta(x, y)$
  - Find the best fit, in the sense of minimizing the squared error, using standard linear regression

Source: RN Figure 17.2

7

## Approximating Monte Carlo Learning

- For MC learning we get a set of training samples  $((x_1, y_1), u_1), ((x_2, y_2), u_2), \dots, ((x_n, y_n), u_n)$   
Where  $u_j$  is the measured utility of the  $j^{\text{th}}$  example - observed total reward from state  $s$  onward in the  $j^{\text{th}}$  trial
- This is a supervised learning problem
  - Standard linear regression problem with squared error and linear function
  - Minimize squared-error (loss) function - when partial derivatives wrt to coefficients of linear function are zero

8

## Online Learning

- To minimize the squared error using online learning
  - Update the parameters after each trial.
- For the  $j^{\text{th}}$  example, take a step in the direction of the gradient of error function:
 
$$\mathcal{E}_j(s) = \frac{(\hat{U}_\theta(s) - u_j(s))^2}{2}$$

Half the squared difference of predicted total and actual total

Widrow-Hoff Rule Or Delta rule for online least squares
- For parameter  $\theta_i$ :  

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \mathcal{E}_j(s)}{\partial \theta_i} = \theta_i + \alpha (u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

Rate of change of error w.r.t each parameter  $\theta_i$
- Notes:
  - Changing the parameters  $\theta_i$  in response to an observed transition between two states also changes the values of  $\hat{U}_\theta$  for every other state!  $\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$
  - Function approximation allows a reinforcement learner to generalize from its experiences.

9

## Example: Navigation in Grid World

- Use:  

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$
  - If  $(\theta_0, \theta_1, \theta_2) = (0.5, 0.2, 0.1)$ , then  $\hat{U}(1,1) = 0.8$
- After a single trial:
  - Suppose we run a trial and the total reward obtained starting at (1,1) is 0.4.
  - Apply delta rule for online least squares to the example where  $\hat{U}_\theta(x, y) = 0.8$  and  $u_j(1,1) = 0.4$ .
  - Parameters  $\theta_0$ ,  $\theta_1$ , and  $\theta_2$  are all decreased by  $0.4\alpha$ , which reduces the error for (1,1).
- Applying Delta Rule for linear function approximator:
 
$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \mathcal{E}_j(s)}{\partial \theta_i} = \theta_i + \alpha (u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

$$\theta_0 \leftarrow \theta_0 + \alpha (u_j(s) - \hat{U}_\theta(s))$$

$$\theta_1 \leftarrow \theta_1 + \alpha (u_j(s) - \hat{U}_\theta(s)) x$$

$$\theta_2 \leftarrow \theta_2 + \alpha (u_j(s) - \hat{U}_\theta(s)) y$$

10

## Approximating Temporal Difference Learning

- For TD learning the same idea of online learning can be applied
  - Adjust the parameters to reduce the temporal difference between successive states
- For utilities:  

$$\theta_i \leftarrow \theta_i + \alpha [R(s, a, s') + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

Update parameter to reduce temporal difference
- For Q-learning:  

$$\theta_i \leftarrow \theta_i + \alpha [R(s, a, s') + \gamma \max_a \hat{Q}_\theta(s', a) - \hat{Q}_\theta(s, a)] \frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i}$$

TD target
- Notes:
  - Also called semi-gradient as the target is not a true value, depends on  $\theta$
  - For passive TD learning, update rule converges for linear function when using on-policy

11

## The Deadly Triad

- Challenges for active learning and non-linear functions:
  - Instability and divergence may arise when following 3 elements are combined:
- Function approximation:
  - Required when state space is large, e.g., using linear function approximation or deep neural nets
- Bootstrapping:
  - Using existing estimates as targets, e.g., in TD, rather than complete returns like in MC methods
- Off-policy training:
  - Training on transitions other than those produced by the target policy

12

## Example: Baird's Counterexample

- Off-policy TD with linear function approximation diverges

Source: SB Figure 11.1

Semi-gradient Off-policy TD

$r(s, a, s') = 0$

$\partial_{\theta}(s) = 0$

$\pi(s) = 1, 1, 1, 1, 1, 1, 1, 10, 1 >^T$

$b(s) = 6/7$

$b(s') = 1/7$

$\gamma = 0.99$

Source: SB Figure 11.2

13

## Catastrophic Forgetting

- Problems with over-training
  - Forgotten about the “dangerous zones” of the learning regions
- Potential solution: Experience replay
  - Retain “relevant” training examples or trajectories from entire learning process
  - Replay those trajectories to ensure utility or value function is still accurate for parts of state space it no longer visits

14

## Non-Linear Functional Approximation

Deep reinforcement learning

15

## Deep Reinforcement Learning

- Deep neural network in function approximation
  - Discovers useful features by itself
  - “Transparent” to show selected features if last network layer is linear
  - $\hat{U}_{\theta}(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \dots + \theta_n f_n(s)$
- Parameters are all weights in all the layers of the network
- Gradients required the same for supervised learning, computed by back propagation

16

## Deep Reinforcement Learning

- Learning the parameters:
  - For utilities:
$$\theta_i \leftarrow \theta_i + \alpha [R(s, a, s') + \gamma \hat{U}_{\theta}(s') - \hat{U}_{\theta}(s)] \frac{\partial \hat{U}_{\theta}(s)}{\partial \theta_i}$$
- For Q-learning:

$$\theta_i \leftarrow \theta_i + \alpha [R(s, a, s') + \gamma \max_a \hat{Q}_{\theta}(s', a') - \hat{Q}_{\theta}(s, a)] \frac{\partial \hat{Q}_{\theta}(s, a)}{\partial \theta_i}$$

17

## Deep Q-Network (DQN)

- Uses deep neural networks with Q-learning to play 49 Atari games
  - Online Q-learning with non-linear function approximators is unstable and may diverge
- DQN uses experience replay with fixed Q-targets:
  - Take action  $a_t$  using  $\epsilon$ -greedy policy
  - Store  $(s_t, a_t, r_{t+1}, s_{t+1})$  in a large buffer  $D$  of most recent transitions
  - Sample a random mini-batch  $(s, a, r, s')$  from  $D$
  - Set targets to  $t + \gamma \max_a Q(s', a', \theta^*)$
  - Do gradient step on the minibatch squared loss w.r.t.  $\theta$  – Optimize MSE btw Q-network and Q-learning targets:
$$L(\theta_t) = E_{s, a, r, s' \sim D} [(r(s, a, s') + \gamma \max_a Q(s', a'; \theta^*)) - Q(s, a; \theta_t)]^2$$
- Experience replay and fixed target
  - Help reduce instability by making input less correlated

18

## Deep Reinforcement Learning: Reality

“

Despite its impressive successes, deep RL still faces significant obstacles: it is often difficult to get good performance and the trained system may behave very unpredictably if the environment differs even a little from the training data.

Deep RL is rarely applied in commercial settings. It is, nonetheless, a very active area of research.

”

Russell, Stuart; Norvig, Peter. Artificial Intelligence (Pearson Series in Artificial Intelligence) (p. 858). Pearson Education. 4<sup>th</sup> Edition.

19

## “Real World” Reinforcement Learning

NUS Computing

CS4246/CS5446  
AI Planning and Decision Making

20

## Policy Search

Policy gradient, actor critic, correlated sampling

21

## Policy Search

- A policy  $\pi: S \rightarrow A$  is a mapping from state to action
- Assume the policy is parametrized by some parameters  $\theta$ 
  - Dimensions of  $\theta$  should be smaller than the number of states
- Often use:  $Q$ -function parameterized by  $\theta$  to represent  $\pi$   

$$\pi(s) = \arg \max_a \hat{Q}_\theta(s, a)$$

What problem can this representation pose when trying to optimize the policy?
- Policy search adjusts  $\theta$  to improve the policy
- Idea:
  - Keep *twiddling* the policy as long as its performance improves, then stop

22

## Intuition

- Policy search tries to find a good policy, e.g., represented as  $Q$ -function
  - Results in process that learns  $Q$ -functions
  - **Q-learning with function approximation:** find a value of  $\theta$  such that  $\hat{Q}_\theta$  is close to  $Q^*$ , the optimal  $Q$ -function
  - **Policy search:** find a value of  $\theta$  that results in good policy
- **Difference between good  $Q$ -function and optimal  $Q$ -function**
  - Approximate  $Q$ -function defined by  $\hat{Q}_\theta = \frac{Q^*}{100}$  gives optimal performance, even though it is not at all close to  $Q^*$

23

## Stochastic Policy

- For policy representation of the form:  

$$\pi(s) = \arg \max_a \hat{Q}_\theta(s, a)$$
- Problem:
  - When actions are discrete, policy is a discontinuous function of  $\theta$
  - This makes gradient-based search difficult
- Stochastic policy:
  - $\pi_\theta(s, a)$  specifies the probability of selecting an action  $a$  in state  $s$
  - E.g., Softmax function, with  $\beta > 0$  modulating softness of the softmax:

$$\pi_\theta(s, a) = \frac{e^{\beta Q_\theta(s, a)}}{\sum_{a'} e^{\beta Q_\theta(s, a')}}$$

24

## How to Improve Policy?

- Definition:**
  - Let  $\rho(\theta)$  be the **policy value** – expected reward-to-go when  $\pi_\theta$  is executed.
- For deterministic policy and deterministic environment:**
  - If  $\rho(\theta)$  is differentiable:
    - Take a step in the direction of the **policy gradient** vector  $\nabla_\theta \rho(\theta)$  – Look for the local optimum
- For stochastic environment and/or policy  $\pi_\theta(s, a)$ :**
  - Obtain an unbiased estimate of the gradient at  $\theta$ ,  $\nabla_\theta \rho(\theta)$  directly from results of trials executed at  $\theta$

25

## Policy Gradient

- Consider: single action from single state  $s_0$   
 $\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a R(s_0, a, s_0, \pi_\theta(s_0, a)) = \sum_a R(s_0, a, s_0) \nabla_\theta \pi_\theta(s, a)$
- Approximate the summation using samples generated from  $\pi_\theta(s_0, a)$ :  
 $\nabla_\theta \rho(\theta) = \sum_a \pi_\theta(s_0, a) \cdot \frac{R(s_0, a, s_0) \nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s_0, a)} \approx \frac{1}{N} \sum_{j=1}^N \frac{R(s_0, a_j, s_0) \nabla_\theta \pi_\theta(s_0, a_j)}{\pi_\theta(s_0, a_j)}$
- For sequential case, this generalizes to:  
 $\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N \frac{u_j(s) \nabla_\theta \pi_\theta(s, a_j)}{\pi_\theta(s, a_j)}$

Sample using policy  
for each state  $s$  visited, where  $a_j$  is executed in  $s$  on the  $j$ th trial and  $u_j(s)$  is the total reward received from state  $s$  onward in the  $j$ th trial.

26

## Derivation: REINFORCE\*<sup>1</sup>

- Alternately, for sequential case, policy gradient by sample approximation generalizes to  

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_\tau p_\theta(\tau) u(\tau)$$

We want to find the  $\theta$  that maximizes the value of  $\sum_\tau p_\theta(\tau) u(\tau)$
- Where,  $\tau$  is the trajectory generated by the policy and  $u(\tau)$  is the sum of rewards from trajectory  $\tau$
- Using the **policy gradient theorem**<sup>1</sup> this can be written as:  

$$\nabla_\theta \rho(\theta) \propto \sum_s p_{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) \hat{Q}_{\pi_\theta}(s, a)$$

States generated by policy      Q-function at that state  
 Gradient of policy      Sutton & Barto Section 13.2

27

## Derivation: REINFORCE\*<sup>2</sup>

- We can approximate the gradient using sampling  

$$\nabla_\theta \rho(\theta) \propto \sum_s p_{\pi_\theta}(s) \sum_a \frac{\pi_\theta(s, a) \nabla_\theta \pi_\theta(s, a) \hat{Q}_{\pi_\theta}(s, a)}{\pi_\theta(s, a)}$$

$$\approx \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^{n_i} \frac{\nabla_\theta \pi_\theta(s_{ij}, a_{ij}) u_i(s_{ij})}{\pi_\theta(s_{ij}, a_{ij})}$$

i: trial, j: seq within trial
- Where,  $a_{ij}$  is executed in  $s_{ij}$  on the  $j$ th step of the  $i$ th trial and  $u_i(s_{ij})$  is the total reward (return) received from the  $j$ th step onward in the  $i$ th trial

Sample using policy  
28

## REINFORCE

- Using an online update, we get the REINFORCE algorithm:  

$$\theta_{j+1} = \theta_j + \alpha u_j \frac{\nabla_\theta \pi_\theta(s, a_j)}{\pi_\theta(s, a_j)}$$
- Using the identity:  

$$\nabla_\theta \ln \pi_\theta(s, a_j) = \frac{\nabla_\theta \pi_\theta(s, a_j)}{\pi_\theta(s, a_j)}$$
- Rewriting:  

$$\theta_{j+1} = \theta_j + \alpha u_j \nabla_\theta \ln \pi_\theta(s, a_j)$$

Ref: SB Section 13.3  
Original ref: Williams, R. 1992  
29

## Variance reduction using a Baseline

- We estimate  

$$\nabla_\theta \rho(\theta) = \sum_s p_{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) \hat{Q}_{\pi_\theta}(s, a)$$

Lower variance
- This is the same as  

$$\sum_s p_{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) (\hat{Q}_{\pi_\theta}(s, a) - B(s))$$

Sum to 1

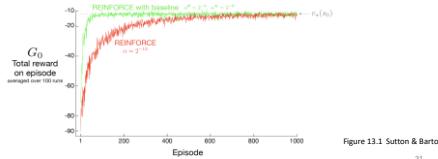
For any function  $B(s)$  because  

$$\sum_a \nabla_\theta \pi_\theta(s, a) B(s) = B(s) \nabla_\theta \sum_a \pi_\theta(s, a) = B(s) \nabla_\theta 1 = 0$$

30

## Variance reduction using a Baseline

- Using a baseline function  $B(s)$  can reduce variance
- Natural choice: estimated  $\hat{U}_{\pi_\theta}(s)$
- The function  $A_{\pi_\theta}(s, a) = Q_{\pi_\theta}(s, a) - \hat{U}_{\pi_\theta}(s)$  is called the **advantage function**



31

## Actor-Critic

- In policy search, a gradient step to update parameters  $w$  (to differentiate from the policy parameters) for the utility (value) function estimator is usually also done
  - To estimate both utility (value) and policy
- This is one form of actor-critic method**
  - Learn a policy (**actor**) that takes action
  - Simultaneously, learn a utility (value) or  $Q$ -function that is used *only* for evaluation (**critic**)

32

## Actor-Critic

- REINFORCE:** Uses a Monte Carlo estimate of the advantage function, which has a higher variance
- For the TD method, the advantage function is:
 
$$\hat{Q}_{\pi_\theta}(s, a) - \hat{U}_{\pi_\theta}(s) = E[r + \gamma \hat{U}_{\pi_\theta}(s')] - \hat{U}_{\pi_\theta}(s)$$
- Using utility (value) function estimator  $\hat{U}(s, w)$  with parameter  $w$ , TD-type update becomes:
 
$$\theta_{j+1} = \theta_j + \alpha \nabla_\theta \ln \pi_\theta(s_j, a_j) \left( r_j + \gamma \hat{U}(s_{j+1}, w) - \hat{U}(s_j, w) \right)$$
advantage
- It is common to use multiple steps of rewards instead of one step:
 
$$r_j + \gamma r_{j+1} + \gamma^2 r_{j+2} + \dots + \gamma^k \hat{U}(s_{j+k}, w)$$

33

## Correlated Sampling

- Improve performance of policy search**
  - Given environment simulator with repeatable random-number sequences
  - Generate a number of experiences in advance, and check the policies with the same set of experiences
  - Eliminate errors due to actual experiences encountered
- Main idea:**
  - No. of random sequences required to ensure value of every policy is well estimated depends only on complexity of policy space, and not on complexity of underlying domain
- Example:**
  - PEGASUS: for stable autonomous helicopter flighted (Ng and Jordan 2000)

34

34

 NUS | Computing

## Partially Observable Markov Decision Process

CS4246/CS5446  
AI Planning and Decision Making

1

### Partially Observable Markov Decision Process (POMDP)

- An POMDP  $M \triangleq (\mathcal{S}, \mathcal{A}, \mathcal{E}, \mathcal{T}, \mathcal{O}, \mathcal{R})$  consists of
  - A set  $\mathcal{S}$  of states
  - A set  $\mathcal{A}$  of actions
  - A set  $\mathcal{E}$  of evidences or observations or percepts
  - A transition function  $\mathcal{T}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$  such that:
 
$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}: \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') = \sum_{s' \in \mathcal{S}} P(s'|s, a) = 1$$
  - An observation function  $\mathcal{O}: \mathcal{S} \times \mathcal{E} \rightarrow [0,1]$  such that:
 
$$\forall s \in \mathcal{S}, \forall e \in \mathcal{E}: \sum_{e' \in \mathcal{E}} \mathcal{O}(s, e) = \sum_{e' \in \mathcal{E}} P(e|s) = 1$$
  - A reward function  $\mathcal{R}: \mathcal{S} \rightarrow \mathbb{R}$  or  $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$
  - Solution: What is a policy in POMDP?

2

### Why Study POMDPs?

- Uncertainty in action outcomes**
  - MDP: fully observable environment
  - Agent knows exactly which state it is in
- Uncertainty in observations**
  - POMDP: partially observable environment
  - Agent does not know exactly which state it is in – cannot observe the state directly
  - Some noisy observations projected from a state
- Real-world challenges**
  - Model sequential decision problem for an uncertain, dynamic, partially observable environment
  - If the state is not directly observable, how does an agent reason about its decisions?

3

### Observation Function and Sensor Model

- Observation function:**
  - $\mathcal{O}(s, e) = P(e|s)$  is the probability of observing (or perceiving evidence)  $e$  from state  $s$
  - Define  $\mathcal{O}(s, e)$  for all  $s \in \mathcal{S}$  and  $e \in \mathcal{E}$
  - Assumption of Markov property
- Sensor model**
  - The observation function defines the sensor model
  - An **observation** is also called a **measurement** or **test**

4

### Belief State as Probability Distribution

- Belief State:**
  - Actual state of the system is unknown, but we can track the probability distribution or **belief state** over the possible states
  - An action  $a$  changes the belief state  $b$ , not just the physical state  $s$
  - $b(s)$  denotes probability assigned to actual state  $s$  by belief state  $b$
  - If  $b(s)$  is the current belief, the agent executes action  $a$  and receives evidence  $e$ , then the updated belief is given by **filtering**:
$$b'(s') = \alpha P(e'|s') \sum_s P(s'|s, a) b(s)$$

(transition model)

where  $\alpha$  is the normalizing constant that makes the belief state sum to 1
- Filtering Function:**

5

### Intuition on Belief State Update

- Recall that Bayes Theorem:**
$$P(A|B) = \alpha P(B|A)P(A)$$

where  $\alpha$  is normalizing constant.

- Then:**
$$\begin{aligned} b'(s') &= P(s'|e', a) && \text{(by definition)} \\ &= \alpha P(e'|s') P(s', a) && \text{(Bayes Theorem)} \\ &= \alpha P(e'|s') \sum_s P(s'|s, a) b(s). && \text{(marginalization over } s\text{)} \end{aligned}$$

6

# Decision Making in POMDPs

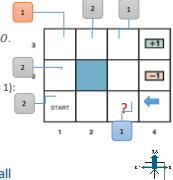
- **Main ideas:**
    - Optimal action depends only on agent's current belief
    - Optimal policy can be described by a mapping  $\pi^*(b)$  from belief to action
    - Optimal policy does not depend on the actual state agent is in!
    - Include **value of information** as a component in decision making
  - **Decision cycle of a POMDP agent:**
    - Given current belief  $b$ , execute action  $a = \pi^*(b)$
    - Receive percept  $e'$
    - Set belief to  $b' = \text{FORWARD}(b, a, e')$  and repeat

7

## Example: Navigation in Grid World

- **POMDP :**
    - States  $S$ , actions  $A$ , transition  $T$ , reward  $R$ , and observation model  $O$ .
  - **Assume:**
    - An observation function measures **no. of adjacent walls** in a state  $s$
    - For all non-terminal states except those in column 3 (where value = 1):  
 $O(s, 2) = 0, O(s, *) = 0.1$  (where \* indicates a wrong value)
  - **Belief state:**
    - $b = (1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 0, 0, 0)$
    - $b(1, 1) = 1/9, b(1, 2) = 1/9, \dots, b(3, 4) = 0$
  - **Suppose agent moves Left and sensor reports 1 adjacent wall**
    - It is quite likely that agent is now in (3, 1)
  - **What is the exact probability values of the new belief state?**

Source: RN Chapter 16



8

8

## Calculating New Belief State

- What is the probability that an agent in belief state  $b$  reaches belief state  $b'$  after executing action  $a$ ?
    - With current belief  $b(s)$ , agent executes action  $a$  and perceives  $e'$ , updated belief is:  

$$b'(s') = \alpha P(e'|s') \sum_s P(s'|s, a) b(s) = \text{FORWARD}(b, a, e')$$
 where  $\alpha$  is normalizing constant for belief state to sum to 1
    - New belief  $b'$  is a conditional probability over actual state given sequence of percepts and actions so far
    - If action and subsequent percept were known, deterministic update to the belief using  $b' = \text{FORWARD}(b, a, e')$
    - But subsequent percept is not yet known, so agent might arrive in one of several possible belief states  $b'$ , depending on percept received

9

9

## Calculating Probability of Percept

- Probability of percept

- Probability of perceiving  $e'$ , given that  $a$  was performed starting in belief state  $b$ , is given by summing over all the actual states  $s'$  that the agent might reach:

$$\begin{aligned}
 P(e'|a, b) &= \sum_{s'} P(e'|a, s', b)P(s'|a, b) \\
 &= \sum_{s'} P(e'|s')P(s'|a, b) \\
 &= \sum_{s'} P(e'|s') \sum_s P(s'|s, a) b(s)
 \end{aligned}$$

Sensor model      Transition model

10

10

## Calculating Transition Model and Reward Model

- **Probability of reaching  $b'$  from  $b$ , given action  $a$ :**
    - Transition model for belief state space:  

$$P(b'|a, b) = \sum_{e'} P(b'|e', a, b) P(e'|a, b)$$

$$= \sum_{e'} P(b'|e', a, b) \sum_{s'} P(e'|s') \sum_s P(s'|s, a) b(s).$$

Senior model
Transition model

where  $P(b'|e', a, b) = 1$  if  $b' = \text{FORWARD}(b, a, e')$  and 0 otherwise
  - **Reward function of belief state space:**
    - Expected reward if the agent does  $a$  in belief state  $b$ :
$$r(b, a) = \sum_b h(c) \sum_s P(s'|s, a) P(c, a, s')$$

11

## Reducing POMDP into an MDP

- Belief space MDP:

- Transition model  $P(b'|b, a)$  and reward model  $\rho(b, a)$  define an observable MDP on the space of belief states!
  - Solving a POMDP on a physical state space – solving a continuous, usually high-dimensional MDP on the corresponding belief-state space
  - An optimal policy for this MDP,  $\pi^*(b)$  is also an optimal policy for the original POMDP

- Remember:

- The belief state is always observable to the agent, by definition

13

12

## Value Iteration

13

## Value Iteration for POMDPs

- Policy and conditional plan**
  - Consider an optimal policy  $\pi^*$  and its application in a specific belief state  $b$
  - Policy generates an action, then for each subsequent observation or percept, belief state is updated and new action is generated, and so on
  - For  $b$ , the policy is equivalent to a **conditional plan**
- Main issue:**
  - How does the expected utility of executing a fixed conditional plan varies with the initial belief state?

14

## Conditional Plan

- A policy at a belief  $b_0$  is a **conditional plan**
- Multiple conditional plans are possible

15

## Utility Function for Belief State

- Note:**
  - A belief state  $b$  is a probability distribution
  - Each utility value in a POMDP is a function of an entire probability distribution
- Problems:**
  - Probability distributions are continuous
  - Huge complexity of belief spaces
- Solution:**
  - For finite state, action, and observation spaces and planning horizon: utility functions represented by piecewise linear functions over belief space

16

## Utility Function of Belief State

- Let the utility of executing a fixed conditional plan  $p$  starting in physical state  $s$  be  $\alpha_p(s)$  – **alpha vector**
- Expected utility of executing  $p$  in belief state  $b$  is:  

$$U_p(b) = \sum_s b(s)\alpha_p(s) \text{ or } b \cdot \alpha_p \text{ (inner product of vectors } b, \alpha_p\text{)}$$
  - A linear function of  $b$ , corresponding to a hyperplane in belief space

17

## Utility Function of Belief State

- At any particular belief state  $b$ , optimal policy is to choose the conditional plan with highest utility:  

$$U^*(b) = U(b) = \max_p b \cdot \alpha_p$$
- Utility or value function  $U(b)$  on belief states, being the maximum of a collection of hyperplanes, will be **piecewise linear and convex**
- For finite depth, there are only a finite number of conditional plans
  - With  $|A|^d$  actions,  $|E|$  observations, there are  $|A|^{O(|E|^{d-1})}$  distinct depth- $d$  plans

18

## Example: Solving a POMDP, $d = 1$

- Consider one-step plans: [Stay] and [Go]
  - Each receives reward for one transition as follows:

$$\begin{aligned} \alpha_{[Stay]}(A) &= 0.9 R(A, Stay, A) + 0.1 R(A, Stay, B) = 0.1 \\ \alpha_{[Stay]}(B) &= 0.1 R(B, Stay, A) + 0.9 R(B, Stay, B) = 0.9 \\ \alpha_{[Go]}(A) &= 0.1 R(A, Go, A) + 0.9 R(A, Go, B) = 0.9 \\ \alpha_{[Go]}(B) &= 0.9 R(B, Go, A) + 0.1 R(B, Go, B). = 0.1 \end{aligned}$$

- Hyperplanes for  $b \cdot \alpha_{[Stay]}$  and  $b \cdot \alpha_{[Go]}$ 
  - Utility or value function: max of the two linear functions
- What is the one-step optimal policy?
  - Optimal policy is to Stay if  $b(B) > 0.5$  and Go otherwise

Source: RN Figure 17.15 (a)

19

## Example: Solving a POMDP, $d = 2$

- Deriving a solution:
  - Obtain utilities  $\alpha_p(s)$  for all the conditional plans  $p$  of depth 1 in each physical state  $s$
  - Compute utilities for conditional plans of depth 2 by considering:
    - each possible first action
    - each possible subsequent percept, and
    - each way of choosing a depth-1 plan to execute for each percept:
      - [Stay; If Percept = A then Stay else Stay]
      - [Stay; If Percept = A then Stay else Go]
      - [Go; If Percept = A then Stay else Stay] ...
  - There are 8 distinct depth-2 plans in all
    - 4 suboptimal and dominated plans – dominated plans are never optimal
    - 4 undominated plans, each optimal in a specific region
  - The regions partition the belief-state space

20

## Example: Utility of Belief-State $b(B)$ : $d = 2$

Utilities of 8 distinct 2-step plans

Utilities of 4 undominated 2-step plans

- Continuous belief space is divided into regions; the regions partition the belief-state space
  - Each region corresponds to a conditional plan that is optimal for that region
- $U(b)$  is piecewise linear and convex

Source: RN Figure 16.15 (b) and (c)

21

## Value Iteration in POMDP

- In general:
  - Let  $p$  be a depth- $d$  conditional plan with initial action  $a$  followed by depth  $(d - 1)$  subplans  $p \cdot e'$ 

$$\alpha_p(s) = \sum_{s'} P(s'|s, a) \underbrace{R(s, a, s')}_{\text{Transition model}} + \gamma \sum_{e'} P(e'|s') \alpha_{p \cdot e'}(s')$$
- Given utility function:
  - Executable policy extracted by:
    - looking at which hyperplane is optimal at any given belief state  $b$
    - executing the first action of the corresponding plan
- POMDP Value Iteration (VI):
  - maintains a collection of undominated plans ( $p$ ) with their utility hyperplanes (**alpha vectors**  $\alpha_p$ )
  - Cf MDP VI computes one utility number,  $U(s)$  for each state  $s$

22

## POMDP Value Iteration Algorithm

```

function POMDP-VALUE-ITERATION(pomdp, ϵ) returns a utility function
inputs: pomdp, a POMDP with states S , actions $A(s)$, transition model $P(s'|s, a)$,
sensor model $P(e|s)$, rewards $R(s, a, s')$, discount γ
 ϵ , the maximum error allowed in the utility of any state
local variables: U , U' , sets of plans p with associated utility vectors α_p
 $U' \leftarrow$ a set containing all one-step plans $[a]$, with $\alpha_{[a]}(s) = \sum_{s'} P(s'|s, a) R(s, a, s')$
repeat
 $U \leftarrow U'$
 $U' \leftarrow$ the set of all plans consisting of an action and, for each possible next percept,
 a plan in U with utility vectors computed according to Equation (16.18)
 $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
until MAX-DIFFERENCE(U , U') $\leq \epsilon(1 - \gamma)/\gamma$
return U

```

Source: RN Figure 16.16

23

## Notes on Value Iteration

- Algorithm's complexity depends primarily on how many plans get generated.
  - Given  $|A|$  actions and  $|E|$  observations, there are  $|A|^{O(|E|^{d-1})}$  distinct depth- $d$  plans.
  - $|A|^n$  conditional plans generated at level  $d + 1$  before elimination, where  $n$  is the no. of conditional plans at level  $d$
  - Elimination of dominated plans to reduce doubly exponential growth
  - P-SPACE hard – very inefficient in practice
  - Approximate methods necessary
- Intermediate belief states have lower value than state  $A$  and state  $B$ 
  - In the intermediate states the agent lacks information needed to choose a good action.
  - Information has value and optimal policies in POMDPs often include information-gathering actions.

24

## Online Methods

Approximate solutions

25

## Scaling POMDP solvers

- POMDP solvers need to solve two problems
  - Belief tracking/filtering
    - Given the history observed, what is the current belief?
  - Planning
    - Given the current belief, what is the optimal action to take?
- To scale up, need to scale up for both problems

26

## Online Agent for POMDP

- Main ideas:
  - Represent transition model and sensor model by a **dynamic decision network (DDN)**
  - Belief tracking – Inference in DDN – Exact inference is computationally intractable
    - No known polynomial time algorithm, as number of state variables grow
  - Approximate solution: Deploy a **filtering** algorithm (exact or particle filtering) to incorporate each new percept and action and to update belief state representation
    - Projecting forward possible action sequences and choosing the best one
- Note:
  - A DDN can actually be used as inputs for any POMDP solution algorithm

27

## Dynamic Decision Network (DDN)

- Execution of POMDP over time can be represented as a DDN
  - Transition and sensor models represented by a Dynamic Bayesian Network (DBN)
  - Add decision and utility nodes to get DDN
- In DDN:
  - State  $S_t$  becomes set of variables  $X_t$
  - Evidence/observation variables are  $E_t$
  - Action at time  $t$  is  $A_t$
  - Transition:  $P(X_{t+1}|X_t, A_t)$
  - Sensor model:  $P(E_t|X_t)$

28

## Look-Ahead Solution of POMDP

$A_t$  in  $P(X_t | E_{1:t})$

$E_{t+1}$

$A_{t+1}$  in  $P(X_{t+1} | E_{1:t+1})$

$E_{t+2}$

$A_{t+2}$  in  $P(X_{t+2} | E_{1:t+2})$

$E_{t+3}$

$U(X_{t+3})$

Belief state

Evidence

Alternate form

Time complexity:  $O(|A|^d |E|^d)$

Source: RN 3e Figure 16.11

29

## Notes on Online Solution

- Main ideas:
  - Project action sequences forward from current belief state
  - Determine utility estimates at projected depth based on future rewards
  - Extract decision by backing up utilities from the leaves
    - Averaging chance nodes and maximizing/minimizing decision nodes
  - Non-leaf states may have rewards
  - Decision nodes correspond to belief states rather than actual states
- Complexity
  - Time complexity of exhaustive search to depth  $d$  is  $O(|A|^d |E|^d)$ 
    - where  $|A|$  is the no. of available actions and  $|E|$  is the number of possible percepts

30

**POMCP<sup>1</sup>**

Partially Observable Monte Carlo Planning

MCTS:

- Select
- Expand
- Simulate
- Backup

- To run UCT on POMDP, we need to represent beliefs in the nodes
  - Inference to construct beliefs is intractable in general
  - For MDP, states are easy to generate
  - Beliefs are difficult to generate, so POMCP uses action-observation history
  - History is equivalent to belief assuming same initial belief;  $|A|^d k^d$
- Instead of propagating beliefs forward in a trial
  - POMCP samples a state at the root from the initial belief.
  - Run the simulation using the state to generate action-observation history
  - Only need to sample from  $P(s'|s, a)$  then  $P(e'|s')$  to generate observation  $e'$  for the action-observation history
  - Avoid constructing beliefs

Silver, D. and J. Veness, Monte-Carlo planning in large POMDPs, in *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010*. 2010. Vancouver, British Columbia, Canada. p. 2164-2172

31

**DESPOT<sup>2</sup>**

Determinized Sparse Partially Observable Tree

Source: Ye et al 2017

- Construct search tree differently! (make tree smaller)
  - Construct  $k$  different search trees
  - Each search tree, sample a state at the root to initialize
  - At every action node, try all actions on the (single) state at that node
  - At every observation node, sample a single observation from the (single) state at node
- Size of the tree:
  - Each of the  $k$  trees have size  $A^d$ . Combine together to get tree of size  $O(|A|^d k)$
  - Exponentially smaller than  $|A|^d |E|^d$
- Can show that searching sampled tree is sufficient if a small good policy exists
  - Use heuristics to do anytime search of this tree

Ye, N., et al., DESPOT: online POMDP planning with regularization. *J. Artif. Int. Res.*, 2017. 58(1): p. 231–266.

32



## Game Theory

CS4246/CS5446  
AI Planning and Decision Making

1

### Game Theory: Overview

- **Multagent decision making**
  - Simultaneous moves and partial observability
  - Perfect information vs imperfect information
- **Agent design**
  - Analyze agent's decisions; compute expected utility for each decision for optimal agents
  - E.g., Two finger Morra
- **Mechanism design**
  - Define rules for the environment with multiple agents to maximize collective good
  - E.g., Protocols for internet traffic routers, intelligent multiagent systems
- **Assumption:**
  - All agents obey optimal decision making rules according to game theory

2

### Elements of A Game

- **Rules of the Game: PAPI**
  - Players
  - Actions
  - Payoffs
  - Information
- **Objective**
  - Describe a situation in terms of the rules of the game
  - Players will maximize payoffs
  - Players will devise strategies to pick actions depending on information available
  - Combination of strategies chosen by each player is the equilibrium
  - Given an equilibrium, predict outcome of the game

3

### Non-Cooperative Games

- **What are non-cooperative games?**
  - Mathematical study of interaction between rational, self-interested agents
- **Why is it called non-cooperative?**
  - While most interested in situations where agents' interests conflict, not restricted to these settings
  - Key is that the agent is the basic modeling unit (with beliefs, preferences, possible actions), and that agents pursue their own interests
- **What is a self-interested agent?**
  - Not that it wants to harm other agents
  - Not that it cares about only things that benefit it
  - Agent has a description of states of the world that it likes; its actions are motivated by this description

4

### Single-Move Games

Normal form games  
Pure strategy games

5

### Single-Move Games

- **Definition:**
  - All players take actions "simultaneously"
  - Result of game is based on this single set of actions
- **Examples:**
  - Prisoner's Dilemma; Two finger Morra; etc.
  - Many real decision making situations in business, politics, defense, financial planning, etc.
- **A single-move game is defined by 3 components**
  1. **Players** or agents who make the decision
    - Focus on 2-player games;  $n$ -player games for  $n > 2$  are also common
  2. **Actions** that players can choose
  3. **Payoff function** that gives the utility to each player for each combination of actions by all players
    - Can be represented as a **payoff matrix** – called the **strategic form** or **normal form**
    - **Row Player**: Player 1; **Column Player**: Player 2

6

## Example: Prisoner's Dilemma

|                    | <i>Al: Testify</i> | <i>Al: Refuse</i> |
|--------------------|--------------------|-------------------|
| <i>Bo: Testify</i> | $B = -5; A = -5$   | $B = 0; A = -10$  |
| <i>Bo: Refuse</i>  | $B = -10; A = 0$   | $B = -1; A = -1$  |

- Game:
  - Two suspects: Bo and Al, are caught near a burglary scene and interrogated separately
  - Choices:
    - One Testifies (blames the other) and the other Refuses (to cooperate): Payoffs = [0, -10] or [-10, 0]
    - Both Refuse: Payoffs = [-1, -1]
    - Both Testify: Payoffs = [-5, -5]
  - Should each of them testify?

7

## Example: Prisoner's Dilemma

- What should Al and Bo do?

|                    | <i>Al: Testify</i>                 | <i>Al: Refuse</i>                  |
|--------------------|------------------------------------|------------------------------------|
| <i>Bo: Testify</i> | $B = \textcolor{red}{-5}; A = -5$  | $B = \textcolor{red}{0}; A = -10$  |
| <i>Bo: Refuse</i>  | $B = \textcolor{blue}{-10}; A = 0$ | $B = \textcolor{blue}{-1}; A = -1$ |

Both should testify!

8

## General Form of Single-Move Games

- Prisoner's dilemma written as a matrix
  - "strategic form" or "normal form"

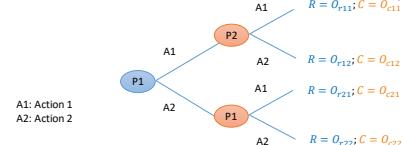
|                                    | <i>Column Player (P2)</i><br>Action 1      | <i>Column Player (P2)</i><br>Action 2 |
|------------------------------------|--------------------------------------------|---------------------------------------|
| <i>Row Player (P1)</i><br>Action 1 | $R = \textcolor{red}{O_{11}}; C = O_{c11}$ | $R = O_{12}; C = O_{c12}$             |
| <i>Row Player (P1)</i><br>Action 2 | $R = O_{r21}; C = O_{c21}$                 | $R = O_{r22}; C = O_{c22}$            |

Subscripts in payoffs denote action numbers of P1 and P2

9

## General Form of Single-Move Games

- Prisoner's dilemma written as a game tree
  - "extensive form"



10

## Game Strategies

- Strategy
  - Each player must adopt and execute a strategy (policy)
- Pure strategy
  - Deterministic policy
  - A single action for a single-move game
- Mixed strategy
  - Randomized policy that selects actions according to a probability distribution
  - $[p; a; (1-p); b]$
  - chooses action a with probability p and b otherwise
- Strategy profile
  - Assignment of a strategy to each player

11

## Game Solutions

- Outcome
  - Given the strategy profile, the game's outcome is a numeric value for each player (number or expected utility)
- Solution
  - A strategy profile in which each player adopts a rational strategy
  - Examples: Pareto optima and Nash equilibrium
- What is "rational"?
  - Each agent chooses only part of the strategy profile that determines the outcome
  - Outcomes are actual results of playing a game
  - Solutions are theoretical constructs for analyzing a game

12

11

12

## Domination

- Dominant strategy**
  - A strategy that dominates all others
  - Irrational to play a dominated strategy
  - Irrational not to play a dominant strategy if one exists
- Strongly dominant strategy**
  - Strategy  $s$  for player P **strongly dominates** strategy  $s'$  if the outcome for  $s$  is better for P than the outcome for  $s'$ , for every choice of strategies by other player(s)
- Weakly dominant strategy**
  - Strategy  $s$  **weakly dominates**  $s'$  if  $s$  is better than  $s'$  on at least one strategy profile and no worse on any other

13

## Deriving Solution with Dominated Strategies

- One way to solve the game: **iteratively eliminate dominated strategies**
  - If we end up with only one strategy for each player, we have a solution!
- Al first reasons that *Refuse* is dominated, so it can be eliminated
- Simplified game after eliminating Al's *Refuse* strategy
- In the simplified game, Bo reasons that *Refuse* is dominated
- On eliminating, we get the solution!

|             | Al: Testify      | Al: Refuse       |
|-------------|------------------|------------------|
| Bo: Testify | $B = -5; A = -5$ | $B = 0; A = -10$ |
| Bo: Refuse  | $B = -10; A = 0$ | $B = -1; A = -1$ |

|             | Al: Testify      |
|-------------|------------------|
| Bo: Testify | $B = -5; A = -5$ |
| Bo: Refuse  | $B = -10; A = 0$ |

|             | Al: Testify      |
|-------------|------------------|
| Bo: Testify | $B = -5; A = -5$ |

14

## Deriving Solution with Dominated Strategies

- Strongly dominant strategy**
  - If iterated elimination of strictly dominated strategies result in one strategy for each player, we have found a unique equilibrium
  - Order of the elimination does not matter
- Weakly dominant strategy**
  - If we also eliminated weakly dominated strategies and end up with one strategy for each player, we have also found an equilibrium, but there may be other equilibria that we did not find

15

## Equilibrium

- Dominant strategy equilibrium**
  - When each player has a dominant strategy, combination of strategies is called a **dominant strategy equilibrium**
  - It is an "equilibrium" because no player has any incentive to deviate from their part of it: by definition, if they did so, they could not do better, and may do worse.
- Nash Equilibrium**
  - A strategy profile forms an equilibrium if no other player can benefit by switching strategies, given that every other player sticks with the same strategy
  - Local optimum in the space of strategies
- Note:**
  - Every game has at least one equilibrium (John Nash)
  - A dominant strategy equilibrium is a Nash equilibrium
  - Some games have Nash equilibrium but no dominant strategies

E.g., in Prisoner's Dilemma, both players testifying  
16

## Pareto Optimality

- An outcome is **Pareto dominated** by another outcome if all players would prefer the other outcome
- An outcome is **Pareto optimal** if there is no other outcomes that all players would prefer, i.e., if there is no other outcome that would make one player better off without making someone else worse off
  - An outcome is Pareto-optimal if there is no other outcome that Pareto-dominates it.
  - A game can have more than one Pareto-optimal outcome
  - Every game has at least one Pareto-optimal outcome

Note: Pareto optimality is named for the Italian economist Vilfredo Pareto (1848–1923).  
17

## Example: Prisoner's Dilemma

|                | Equilibrium | Al: Testify      | Al: Refuse       |
|----------------|-------------|------------------|------------------|
| Pareto optimal | Bo: Testify | $B = -5; A = -5$ | $B = 0; A = -10$ |
|                | Bo: Refuse  | $B = -10; A = 0$ | $B = -1; A = -1$ |

- Dilemma:**
  - Equilibrium outcome of both Al & Bo testifying is worse than the outcome both would get if they refuse
  - The (*Testify, Testify*) equilibrium is **pareto dominated** by (*Refuse, Refuse*) strategy profile which is better for both players
  - Although (*Refuse, Refuse*) combination is better, both players playing **rationally** end up in (*Testify, Testify*) solution

Pareto optimal  
Pareto dominating  
18

## Example: Finding Pure Strategy Nash Equilibrium

|              | Acme: bluray     | Acme: dvd        |
|--------------|------------------|------------------|
| Best: bluray | $B = +9; A = +9$ | $B = -1; A = -4$ |
| Best: dvd    | $B = -1; A = -3$ | $B = +5; A = +5$ |

- Game: (Classic)

- Two video game console manufacturers, Acme and Best, need to decide whether to use Blu-ray discs or DVD
- Profits are positive if they both agree to use the same format, but negative otherwise
- There is no dominant strategy, so iterative elimination of dominant strategies will fail to find a solution.

- How do we find the solution(s) in this case?

19

## Finding Pure Strategy Nash Equilibrium

- A simple algorithm to find pure Nash equilibrium:

- For each column, mark cell if it has the maximum payoff for the row player
    - May be more than one
  - For each row, mark cell if it has the maximum payoff for the column player
    - May be more than one
  - Cells with both row and column marks are pure Nash equilibrium
- There are multiple acceptable solutions, but if agents disagree, both suffer
  - There are various ways the agents can coordinate in choosing solutions
    - e.g., by communicating & negotiation
    - Games where agents need to communicate are called coordination games

20

20

19

## Mixed Strategies Games

Probabilistic action selection with randomized policy

21

## Example: Game of Chicken

|                    | Player A: continue | Player A: swerve |
|--------------------|--------------------|------------------|
| Player B: continue | $B = -10; A = -10$ | $B = -2; A = -2$ |
| Player B: swerve   | $B = -2; A = 2$    | $B = 0; A = 0$   |

- There are two pure strategy Nash equilibria

- (A: continue, B: swerve)
- (A: swerve, B: continue)

- But there is also a mixed strategy that we will compute

23

## Example: Game of Chicken

| [p;<br>1-p]        | Player A: continue | Player A: swerve |
|--------------------|--------------------|------------------|
| Player B: continue | $B = -10; A = -10$ | $B = -2; A = -2$ |
| Player B: swerve   | $B = -2; A = 2$    | $B = 0; A = 0$   |

- B plays continue:

- Let B play continue with probability  $p$  and swerve with probability  $(1 - p)$
- If A plays continue then the expected payoff for A is:  

$$-10p + 2(1 - p) = -12p + 2$$
- If A plays swerve then the expected payoff for A is:  

$$-2p + 0(1 - p) = -2p$$
- It would not matter which action A plays if both the expected payoffs are the same; i.e.,  

$$-12p + 2 = -2p \Rightarrow p = 1/5$$

24

24

23

## Example: Game of Chicken

|                    | Player A: continue | Player A: swerve |
|--------------------|--------------------|------------------|
| Player B: continue | $B = -10; A = -10$ | $B = 2; A = -2$  |
| Player B: swerve   | $B = -2; A = 2$    | $B = 0; A = 0$   |

- A plays *continue*:

- Let A play *continue* with probability  $q$  and swerve with probability  $(1 - q)$
- If B plays *continue* then the expected payoff for B is:  
 $-10q + 2(1 - q) = -12q + 2$
- If B plays *swerve* then the expected payoff for B is:  
 $-2q + 0(1 - q) = -2q$
- It would not matter which action B plays if both the expected payoffs are the same; i.e.,  
 $-12q + 2 = -2q \Rightarrow q = 1/5$

25

## Example: Game of Chicken

|                    | Player A: continue  | Player A: swerve        |
|--------------------|---------------------|-------------------------|
| Player B: continue | $B = -10; p = 1/25$ | $B = 2; (1 - p) = 4/25$ |
| Player B: swerve   | $B = -2; A = 2$     | $B = 0; A = 0$          |

$$p = q = \frac{1}{5}$$

Probabilities of the pairs of actions

### Finding mixed strategy:

- For player B the expected utility is:

$$\frac{1}{25} \times -10 + \frac{4}{25} \times 2 + \frac{4}{25} \times -2 + \frac{16}{25} \times 0 = -\frac{2}{5}$$

- For player A the expected utility is:

$$\frac{1}{25} \times -10 + \frac{4}{25} \times -2 + \frac{4}{25} \times 2 + \frac{16}{25} \times 0 = -\frac{2}{5}$$

Expected payoff for each player for this strategy profile

26

## Finding Mixed Strategies

- Algorithm finds a mixture for one player where the other player becomes indifferent to the choices; repeat this for both players
  - Does not always work! (If there is no mixture where the player becomes indifferent)
  - E.g., if one choice dominates the other choice, the player cannot be indifferent to the choices
- Questions:
  - Is there an efficient algorithm for computing a Nash equilibrium in general?

27

## Interpreting Mixed Strategy Equilibria

- What does it mean to play a mixed strategy? (Different interpretations)

- Randomize to confuse your opponent
  - E.g., Matching pennies
- Players randomize when they are uncertain about the other's action
  - E.g., Battle of the sexes
- Mixed strategies are a concise description of what might happen in repeated play
  - Count of pure strategies in the limit
- Mixed strategies describe population dynamics
  - 2 agents randomly chosen from a population, all having deterministic strategies. Mixed strategy is the probability of getting an agent who will play one pure strategy or another.

28

## Zero-Sum Games

Minimax Theorem

29

## Example: Two-Finger Morra

|        | O: one           | O: two           |
|--------|------------------|------------------|
| E: one | $E = +2; O = -2$ | $E = -3; O = +3$ |
| E: two | $E = -3; O = +3$ | $E = +4; O = -4$ |

Efficient algorithm to compute the equilibrium is known

- Two players: E and O simultaneously display one or two fingers

- Let, total number of fingers be  $f \in \{1, 2, 3, 4\}$ :

- If  $f$  is odd, O collects \$f from E
- If  $f$  is even, E collects \$f from O

- What is the optimal mixed strategy against a rational player?

- What is the expected utility for each player?

- Special case of two player zero-sum games: sum of the payoffs is always zero

- Only need to consider payoff of one player, since the sum is 0: Use the payoff of the even player, E
  - E will try to maximize the payoff  $U_E(e, o)$ ; O will try to minimize the payoff

30

30

### Two Finger Morra: Minimax Game Tree

- Find range of true utility  $U$  of solution**
  - Examining pure strategies:
$$U_{E,O} \leq U \leq U_{O,E}: -3 \leq U \leq 2$$

E picks action first & reveals it to O

Source: RN Figure 17.12 (a) and (b)

31

### Two Finger Morra: Calculating Utility

- For E: 

|     |       |       |
|-----|-------|-------|
|     | one   | two   |
| One | (2)-2 | (3)+3 |
| Two | (3)+3 | (4)-4 |

$$\{p: one; (1-p): two\}$$
- O picks action first & reveals it to E
- If O chooses one: expected payoff to E:  $2p - 3(1-p) = 5p - 3$   
If O chooses two: expected payoff to E:  $-3p + 4(1-p) = 4 - 7p$
- Draw straight lines in graph (e)
  - O, the minimizer, will always choose lower of the two lines
  - The best E can do at the root is to choose  $p$  at intersection pt:  
 $5p - 3 = 4 - 7p \Rightarrow p = 7/12$  and hence  $U_{E,O} = -1/12$

Source: RN Figure 18.12 (c) and (e)

32

### Two Finger Morra: Parameterized Game Tree

- Determine true value of utility  $U$  of solution
- Examining mixed strategies:
 
$$U_{E,O} = -1/12, U_{O,E} = -1/12: U = -1/12$$

$$U_{E,O} = -1/12, U_{O,E} = -1/12: U = -1/12$$

Source: RN Figure 18.12 (c) and (d)

33

### Two Finger Morra: Utility Profile

- Analyze utility functions to find optimal strategy
  - E always chooses probability parameter for mixed strategy at intersection point

Source: RN Figure 18.12 (e) and (f)

34

### Two Finger Morra: Maximin Equilibrium

- E first: O selects action with smaller expected value, i.e., min of two linear functions.**
  - The best that E can do is to select the value  $p$  that maximized the min of the two functions (maximin strategy)
  - $p = 7/12$
- O first: E finds max of the two actions**
  - The best that O can do is to select the value  $q$  that minimizes the function (minimax strategy)
  - $q = 7/12$
- Note: Max of the first function is the same as min of the second! (A special case)**
- True utility is attained by the mixed strategy, which should be played by both players.**
  - $[7/12: one; 5/12: two]$
- This strategy is called the maximin equilibrium of the game, and is a Nash equilibrium.**

35

### Minimax Theorem (von Neumann 1928)

- In any finite, two-player, zero-sum game, in any Nash equilibrium each player receives a payoff that is equal to both his maximin value and his minimax value
- It follows that:
  - Each player's maximin value is equal to his minimax value. By convention, the maximin value for player 1 is called the value of the game;
  - For both players, the set of maximin strategies coincides with the set of minimax strategies; and
  - Any maximin strategy profile (or minimax strategy profile) is a Nash equilibrium. Furthermore, these are all the Nash equilibria.
  - Consequently, all Nash equilibria have the same payoff vector (namely, those in which player 1 gets the value of the game)

Source: [SLB] 3.4.1

36

## Complexity of Computing Nash Equilibrium

- Every finite game is known to have at least one Nash equilibrium
  - For 2-player zero sum game, we know how to find NE efficiently
- However, the problem of finding the NE in the general case is believed to be computationally intractable
  - It is PPAD-complete<sup>1</sup>
    - No known efficient algorithm in general
    - Computationally hard even for 2 player games

<sup>1</sup>[https://en.wikipedia.org/wiki/PPAD\\_\(complexity\)](https://en.wikipedia.org/wiki/PPAD_(complexity))



## Human Factors

### Judgmental Biases and Uncertain Preferences

CS4246/CS5446  
AI Planning and Decision Making

1

## Types of Decision Theory

- Normative decision theory
  - Describes how ideal, rational agents should behave
    - Explains how a rational agent should make decisions
    - Proposes models of preference and rational decision making
    - Preference ordering, maximum expected utility
- Descriptive decision theory
  - Describes how actual agents (humans) really behave
    - Explains how we make decisions
    - Identify potential pitfalls and biases
    - Prospect Theory, Naturalistic decision making
- Prescriptive decision theory
  - Prescribes guidelines for agents to behave rationally
    - Explains how we should make rational decisions
    - Provides guidelines and tools to support decision making
    - Decision analysis and modeling

2

## Bounded Rationality

- Views individuals as attempting to make rational decisions
- Acknowledges problem solvers:
  - often lack important information on problem situation
  - faces time and cost constraints that limit quality and quantity of available information;
  - have limited memory capacity; and
  - are limited by intelligence and perceptions to accurately calculate optimal choice
- In real life, problem solvers **satisfice** rather than optimize
  - Forego the best solution in favor of one that is acceptable or reasonable
  - Do not examine all alternatives
  - Search for a solution that meets an acceptable level of performance

3

## Judgmental Heuristics

Power of being human ...

4

## Judgment

- What is judgment?
  - Major class of **cognitive** activity, sf. learning and perception
  - An opinion about what is or will be the status of some aspects of the world
- Types of judgment:
  - Prediction
  - Evaluation
  - Determination
  - Likelihood
- Accuracy:
  - Depends on extent to which the mind mirrors environment it attempts to predict

5

## Judgmental Heuristics

- What are heuristics?
  - **Simplifying strategies** or rules of thumb
  - Simple ways of dealing with complex world
  - Usually provide acceptable results
- Common judgmental heuristics
  - Availability
  - Representativeness
  - Anchoring and adjustments
- Pitfalls:
  - Adopt heuristics without being aware of them
  - Misapplication to inappropriate situations leading to wrong/bad solutions

6

## The Availability Heuristic

- Assess likelihood of event by degree to which instances of event are readily "available" in memory
- Examples:
  - Your Assignment 3 is due in 5 days time. Do you think you will actually finish the assignment by the target date?
  - When was the last time you talked to your boyfriend/girlfriend?
  - When was the last time you went to the post office?
  - In an arbitrary English word, is it more likely to find the letter R in the first position or in the third position?
- Questions:
  - Why is it a good heuristic?
  - Why is it fallible?

7

## The Representativeness Heuristic

- Assess event likelihood by similarity to stereotypes
- Examples:
  - How do you classify a new plant?
  - My friend C is outgoing, outspoken, adventurous, and hopes to become an Air-Force pilot. Do you think C is a man or a woman?
  - What is the likely diagnosis of a patient with fever, running nose, cough and sore throat?
- Questions:
  - Why is it a good heuristic?
  - Why is it fallible?

8

## Anchoring and Adjustment

- Starting from initial value, adjusting to yield final decision
- Examples:
  - If you had score full marks in Test 1 of a difficult module, would you expect to do as well in Test 2 and Test 3 of the same module?
  - If you had a good meal at a new restaurant, would you go back again?
  - "Love at first sight"
- Questions:
  - Why is it a good heuristic?
  - Why is it fallible?

9

## Cognitive Biases and Psychological Traps

Danger zones in decision making and how to avoid them

10

## Hidden Traps in Decision Making

- Recallability trap
- Anchoring trap
- Confirming evidence trap
- Framing trap
- Overconfidence trap and Prudence trap
- Insensitive to sample size trap
- Status-quo trap
- ...

11

## List of Cognitive Biases

- Biases Emanating from the Availability Heuristic
  - 1. Ease of recall
  - 2. Retrievability
  - 3. Presumed associations
- Biases Emanating from the Representativeness Heuristic
  - 4. Insensitivity to base rates
  - 5. Insensitivity to sample size
  - 6. Misconceptions of chance
  - 7. Regression to the mean
  - 8. The conjunction fallacy
- Biases Emanating from Anchoring and Availability
  - 9. Insufficient anchor adjustment
  - 10. Conjunctive and disjunctive events bias
  - 11. Overconfidence
- Two More General Biases
  - 12. The confirmation trap
  - 13. Hindsight and the curse of knowledge
- Others ...

12

## 1. Recallability trap

- What is the trap?**
  - Giving undue weight to recent, dramatic events
  - Based upon vividness and recency
  - An event whose instances are more easily recalled will appear more numerous than an event of equal frequency whose instances are less easily recalled
- What can you do about it?**
  - Carefully examine assumptions to ensure they are not overly influenced by recent memory
  - Get actual statistics if possible
  - Do not be guided by impressions

13

## 9. Anchoring trap

- What is the trap?**
  - Giving disproportionate weight to the initial information received
  - Adjustments away from anchors usually not sufficient to change effects of anchor
  - Answers are biased toward the initial anchor, even if it is irrelevant
  - Different starting points yield different answers
- What can you do about it?**
  - View problem from different perspectives
  - Be open-minded
  - Think of problem first before consulting others
  - Be careful to avoid anchoring others' opinions
  - Be wary of anchors in discussions and negotiations

14

## Example

- You heard that when this great surgeon predicts a success in the operation, the operation is always successful. You are to check this claim.
- Consider the information available about the following four events (cards):

|                                                   |                                                     |                                                 |                                               |
|---------------------------------------------------|-----------------------------------------------------|-------------------------------------------------|-----------------------------------------------|
| Card 1<br>(Prediction)<br><i>Favorable Report</i> | Card 2<br>(Prediction)<br><i>Unfavorable Report</i> | Card 3<br>(Outcome)<br><i>Operation Success</i> | Card 4<br>(Outcome)<br><i>Operation Fails</i> |
|---------------------------------------------------|-----------------------------------------------------|-------------------------------------------------|-----------------------------------------------|

You currently see the predictions (cards 1 and 2) or outcomes (cards 3 and 4) associated with the four events. You are seeing one side of a card. On the other side of cards 1 and 2 is the actual outcome, while on the other side of cards 3 and 4 is the prediction that the great surgeon made.

Which cards would you turn over for the evidence that you need to check the great surgeon's claim?

15

## 12. Confirming evidence trap

- What is the trap?**
  - Seeking out information supporting an existing prediction and to discount opposing information
  - Seek confirmatory evidence and do not search for disconfirming information for decision processes
  - Search for challenging, or disconfirming, evidence will actually provide the most useful insights
- What can you do about it?**
  - Always examine all evidence with equal rigor
  - Get devil's advocate and build counterarguments
  - Be honest about the motives
  - When seeking advice, do not ask leading questions that would lead to confirming evidence

16

## Framing trap

- What is the trap?**
  - Occurs when a problem is misstated, undermining entire decision making process
  - Presentation of information** can significantly impact decision making under uncertainty
  - Intuitions about risk or uncertainty routinely deviate from rationality because of misunderstanding of
    - nature of uncertainty; and
    - effects of framing
  - Often closely related to other traps; a frame can:
    - establish the status quo or introduce an anchor
    - highlight sunk cost or lead toward confirming evidence

17

## Prospect Theory

The graph shows the characteristic S-shape of the prospect theory utility function, which is concave for gains and convex for losses. This reflects the concept of loss aversion.

- Reference- or framing-dependent utility functions
- Changes, not levels, in measures of "goodness" matter
- Gains are different from losses
  - Loss aversion
- Risk preferences
  - Risk aversion over gains
  - Risk seeking over losses
- Diminishing sensitivity
  - Change impact diminishes wrt distance from reference point
- Probability weighting
  - Overweighting lower probability
  - Underweighting medium to high probability

Source: RN Figure 15.2b  
Kahneman, D. and A. Tversky, *Prospect Theory: An Analysis of Decision under Risk*. *Econometrica*, 1979. 47(2): p. 263-291.

18

## In general ...

- Judgmental or behavioral decision making**
  - Important, emerging field, complementing evidence based and classic utility-based decision making
  - Understanding human behavior from systematic studies and experiments and theoretical frameworks
  - Applications in prediction, diagnosis, intervention, management, policy making, and public communication (s.f. Behavioral Economics)
  - Directly relevant to the study of ethical, governance and regulatory considerations in AI
- To improve decision making capabilities (of human-aware AI):**
  - Design agent interaction models considering human judgmental heuristics and cognitive biases
  - Identify common judgmental heuristics
  - Understand potential adverse effects and biases
  - Selectively and correctly apply heuristics and insights

19

## Unknown Preferences

Uncertainty about preferences

20

## Uncertain About Own Preferences

- What if decision maker (agent) does not know about his/her/its own preferences?**
- Example:**
  - You (Agent) want to spend the limited money on ice cream.
  - There are 2 ice cream flavors – Vanilla and Durian
  - You like Vanilla, not sure about Durian
  - How would you choose?
- To handle uncertain preferences in a decision model:**
  - Add a new **latent variable** to represent the unknown preferences
  - Add an appropriate sensor model

21

## Uncertain About Own Preferences

Source: RN Figure 15.10(a) A decision network for the ice cream choice with an uncertain utility function. (b) The network with the expected utility of each action. (c) Moving the uncertainty from the utility function into a new random variable.

(a)

| D/V     | Like/Durian | U      |
|---------|-------------|--------|
| durian  | true        | -\\$82 |
| vanilla | false       | +\\$1  |

(b)

| D/V     | U     |
|---------|-------|
| durian  | +\\$8 |
| vanilla | +\\$1 |

(c)

| D/V     | Like/Durian | U      |
|---------|-------------|--------|
| durian  | true        | -\\$82 |
| vanilla | false       | +\\$1  |
| durian  | false       | +\\$1  |
| vanilla | true        | -\\$82 |

22

## Uncertain About Others' Preferences

- When to defer to humans (or other agents)?**
- Example:**
  - A robot with uncertainty about human preferences will defer to the human and allow itself to be switched off
  - The off-switch game (RN 15.7.2)<sup>1,2</sup>
- Example:**
  - A robot working to assist human with observed preferences to maximize human's payoff
  - The assistance game (RN 17.2.5)<sup>1,2</sup>
  - A formal model of provably beneficial AI**

[1] Hadfield-Menell, D., et al. (2017). The off-switch game. In Proceedings of the 26th International Joint Conference on Artificial Intelligence; Melbourne, Australia. AAAI Press. 220-227  
[2] Russell, S. (2021). *Human-compatible artificial intelligence*. Human-like machine intelligence. S. Muggleton and N. Chater, Oxford University Press: 3-23

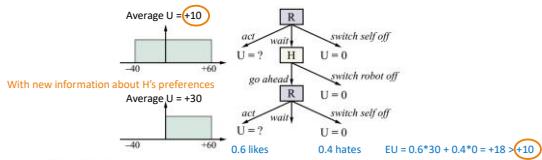
23

## The Off-Switch Game

- A robot with uncertainty about human preferences will defer to the human and allow itself to be switched off.
- Robbie (R) is a software robot assistant working for Harriet (H), a busy human.
- R is helping H make travel arrangement for her business trip; he has 3 choices:
  - R can act now - He is quite unsure how much H will like the arrangement; assume he has a uniform probability for its net value to H between -40 and +60, (average +10)
  - R could "switch himself off—which is (without loss of generality) to have value 0 to H
  - R could explain his plan, wait, and let H switch him off.
    - H can either switch him off or let him go ahead and book the travel.
- What possible good could this do, given that R could make both of those choices himself?**

24

## The Off-Switch Game



Source: RN Figure 16.11

25

## The Off-Switch Game

- Main ideas:
  - $H$ 's choice—to switch  $R$  off or go ahead—provides  $R$  with information about  $H$ 's preferences.
  - Assume  $H$  is rational, so if  $H$  lets  $R$  go ahead, it means the value to  $H$  is positive.
  - As shown in Figure,  $R$ 's belief changes: it is uniform between 0 and +60, with average of +30.
  - For  $R$ 's initial choices from his point of view:
    1. Acting now has an expected value of +10.
    2. Switching himself off has a value of 0.
    3. Waiting and letting  $H$  switch him off leads to two possible outcomes:
      - a) 40% chance, based on  $R$ 's uncertainty about  $H$ 's preferences, that she will switch  $R$  off, with expected value +30.
      - b) 60% chance  $H$  will like the plan and allow  $R$  to go ahead, with expected value +10.
    - Waiting has Expected  $U = (0.4 \times 0) + (0.6 \times 30) = +18 > +10$ .  $R$  expects if he acts now.
    - $R$  has a positive incentive to defer to  $H$ , due to  $R$ 's uncertainty about  $H$ 's preferences.
    - $R$ 's decision provides  $R$  with information, and **expected value of information** is always nonnegative

26

25

26

## The Off-Switch Game

- In general:
  - Formally, let  $P(u)$  be  $R$ 's prior probability density over  $H$ 's utility for the proposed action  $a$ .
  - Then the value of going ahead with  $a$  is
 
$$EU(a) = \int_{-\infty}^0 P(u) \cdot u du = \int_{-\infty}^0 P(u) \cdot u du + \int_0^{\infty} P(u) \cdot u du$$
  - On the other hand, the value of action  $d$ , deferring to  $H$ , is composed of two parts:
    - If  $u > 0$  then  $H$  lets  $R$  go ahead, so the value is  $u$ , but:
    - If  $u < 0$  then  $H$  switches  $R$  off, so the value is 0:
$$EU(d) = \int_{-\infty}^0 P(u) \cdot 0 du + \int_0^{\infty} P(u) \cdot u du$$
  - Hence:
 
$$EU(d) \geq EU(a)$$
  - The two choices have equal value only when the negative region has zero probability—that is, when  $R$  is already certain that  $H$  likes the proposed action.

27

27

## Possible Extensions

- Cost for  $H$ 's Time:
  - To impose a cost for  $H$ 's time:  $R$  is less inclined to bother  $H$  if the downside risk is small
- Probability of Human Error:
  - To allow for probability of human error:  $R$  is less inclined to defer to an irrational  $H$  who sometimes acts against her own best interests
  - Example:
    - If  $R$  is a self-driving car and  $H$  is his naughty two-year-old passenger
    - $R$  should not allow  $H$  to switch him off in the middle of the highway

28

28

## The Assistance Game

- A robot working to assist human with observed preferences to maximize human's payoff
- Robbie ( $R$ ) is a software robot assistant working for Harriet ( $H$ ), a busy human.
- $H$ 's preference parameter  $\theta$  denotes the relative value (in dollars) of a paperclip
  - $H$ 's payoff for  $p$  paperclips and  $s$  staples will be  $p\theta + s(1 - \theta)$  dollars.
  - $R$ 's prior about  $H$ 's preferences is  $P(\theta) = \text{Uniform}(\theta; 0, 1)$ .
- $H$  goes first, and can choose to make two paperclips, two staples, or one of each.
  - Then  $R$  can choose to make 90 paperclips, 90 staples, or 50 of each.

29

29

## The Assistance Game

- At each stage:
  - How does  $R$  learn?
  - How does  $H$  decide?
  - What is the optimal strategy?

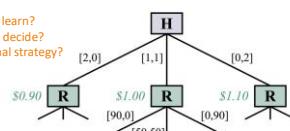


Figure 18.6 The paperclip game. Each branch is labeled  $[p,s]$  denoting the number of paperclips and staples manufactured on that branch. Harriet the human can choose to make two paperclips, two staples, or one of each. (The values in green italics are the values for Harriet if the game ended there, assuming  $\theta=0.45$ .) Robbie the robot then has a choice to make 90 paperclips, 90 staples, or 50 of each.

Source: RN Figure 17.6

30

30

## The Assistance Game

- Note:
  - The off-switch problem is a restricted kind of assistance game
  - General assistance games include actions on:
    - H's part as teaching, rewarding, commanding, correcting, demonstrating, or explaining, and
    - R's part as asking permission, learning from demonstrations, preference elicitation, etc.
  - By solving the game, H and R work out on their own how to convey preference information from H to R, so that R can be more useful to H
- Assume that:
  - Human (H) observes her own preferences  $\theta$  and acts accordingly
  - Robot (R) has a prior probability  $P(\theta)$  over H's preferences.
  - Payoff is defined by  $\theta$  and is identical for both players: both H and R are maximizing H's payoff
  - H has incentive to "signal" to R some information about her preferences.
  - R is able to interpret signal because he can solve the game, and therefore he can understand H's preferences for her to signal in that way.

31

## The Assistance Game

- Provably Beneficial AI:
  - With her strategy, H is teaching R about her preferences using a simple code—a language—that emerges from equilibrium analysis.
  - R never learns H's preferences exactly, but he learns enough to act optimally on her behalf
  - R is **provably beneficial** to H under the assumptions stated, and under the assumption that H is playing the game correctly
- Points to ponder:
  - Would myopic best response works for all cases of assistance games?

32

31

32

## Possible Extensions

- Finding Optimal Strategy Profiles:
  - Provided there are no ties that cause coordination problems, finding an optimal strategy profile for an assistance game is reducible to solving a POMDP whose state space is the underlying state space of the game plus human preference parameters  $\theta$
  - POMDPs that represent assistance games have additional structure that enables more efficient algorithms.
- General Settings:
  - Assistance games can be generalized to allow for: Multiple human participants, multiple robots, imperfectly rational humans, humans who don't know their own preferences, and so on.
- Enhanced Communications:
  - By providing factored or structured action space, opportunities for communication can be greatly enhanced
- Key Property of Assistance Games:
  - The more intelligent the robot, the better the outcome for the human.

33

33