# EKI 2013

## Zusammenfassung

# Intelligent Agents

## Design Principles

### Agents and Environments

Agents: Sensors, Actuators, agent function percepts -> actions (f: P* -> A), agent program produces f

### Rationality

Performance Measure: Criteria of success

Rational: Select action that maximizes performance measure, given percept history and knowledge

### PEAS (Performance Measures, Environment, Actuators, Sensors)

Task environment needs to be specified to design a rational agent

### Environment types

*Fully observable* vs. *partly observable* (can sensors detect relevant properties?)

*Single-agent* vs. *multi-agent* (cooperation/competition)

*Deterministic* vs. *stochastic* (state + action = deterministic state?)

*Episodic* vs. *sequential* (episodic: choice of action depends only on current episode = atomic experience part)

*Static* vs. *dynamic* vs. *semi-dynamic* (world doesn't or does change, or only performance score decreases)

*Discrete* vs. *continuous* (world values, e.g. time)

*Known* vs. *unknown* (knowledge about laws of environment)

### Agent Types

Agent = Architecture (Device, Sensors, Actuators) + Program

Types:

*Simple reflex agents*: Sensors -> Condition-action rules -> Actuators

*Model-based reflex agents*: S ->internal state, evolution of world -> Condition-action rules -> A

*Goal-based agents*: S -> internal & world state ->Effects of actions, Goals -> A

*Utility-based agents*: S -> internal & world state -> happiness/utility function -> A

# Problem Solving and Search

## Uninformed Search

### Single-state problem formulation

Problem Definition:

- Initial state
- Successor function S(x) = set of action-state pairs S(A) = {(A->B, B), … }
- Goal test (e.g. NoDirt(x) , x = A)
- Path cost

Solution: Sequence of actions leading from initial state to a goal state

### State Space

Abstraction of complex real world -> abstract states, actions, solutions

### Tree search algorithms

Simulated exploring of state space, tree of explored states, list of still available states (frontier)

Nodes: state, parent, children, depth, path cost g(x)

### Search strategies

Strategy = order of node expansion

- Completeness (always find existing solution)
- Time complexity (number of nodes expanded)
- Space complexity (maximum number of nodes in memory)
- Optimality (least-cost solution)

Measures: ***b*** =maximum branching factor, ***d*** = depth of least-cost solution, ***m*** = maximum depth

### Types of strategies

- Deterministic search
  - Uninformed (“blind”)
  - Informed / “heuristic”
- Local search
- …

## Uninformed search strategies

### Breadth-first search BFS

Expand shallowest unexpanded node, frontier is FIFO queue

- Complete: if b is finite
- Time: worst case $1 + b + b^2 + \dots + b^d = O(b^d)$
- Space: $O(b^d)$ .. very high amount of data!
- Optimal: only if cost = 1

### Uniform-cost search

Expand least-cost unexpanded node, frontier ordered by path cost

- Complete: if step cost $\geq \varepsilon$

- Time: nodes n with g(n) ≤ C* (optimal solution cost) → $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
- Space: same as time
- Optimal: Yes

### Depth-first search
Expand deepest unexpanded node, frontier is LIFO queue

- Complete: No. Only in finite spaces(depth) without loops
- Time: $O(b^m)$, better than BFS if solutions are dense
- Space: O(bm) linear
- Optimal: No

### Depth-limited search
DFS with depth limit l, nodes at depth l have no successors. Time/Space complexity: replace m with l

### Iterative deepening search
DLS with increasing depth limit l

- Complete: Yes
- Time: $(d+1)b^0 + \ldots + b^d = O(b^d)$
- Space: O(bd)
- Optimal: if step cost = 1

### Repeated States / Graph search
Detect repeated states, add children of expanded node only if not already in the frontier or explored set

## Informed Search
### Heuristic search
- Evaluation function f(n): estimation for a function f*(n)
- Heuristic function h
  - ~ desirability
  - to find most-desired node
  - estimate minimal costs to goal
  - h(n) low computing cost

### Greedy search
f(n) = h(n), expand node with smallest f value

does not take spent cost into account

- Complete: No, Yes with loop-check
- Time: $O(b^m)$
- Space: $O(b^m)$
- Optimal: No

### A*-Search
Evaluation function f(n) = g(n) + h(n)

- g(n) path costs from start to n
- h(n) estimated cost to goal from n
  - hast to be admissible

    - $h(n) \leq h^*(n)$ ~ "optimistic" .. $h^*$ == true cost
    - $h(n) \geq 0$
    - $h(g) = 0$ … for every goal g
- f(n) estimated total cost of path through n to goal

### Optimality of A*

In tree search: optimal because h admissible

Graph search: g(n) is not required to be minimal and more than one path exists → one can reach non-optimal costs.

Solutions:

- Change algorithm, more complicated bookkeeping → affects run-time
- Stronger restriction on heuristic: consistency

Heuristic h is *consistent* if for every node n, operator a, successor n' and path cost c:

$$h(n) \leq c(n, a, n') + h(n')$$

→ f is non-decreasing → A* graph search is optimal

### Properties of A*

- Complete: Yes, unless there are infinite nodes with $f \leq f(G)$
- Time: Exponential in [relative error in h * length of solution]
- Space: Exponential (Keep all nodes)
- Optimal: Yes
- A* expands all nodes with $f(n) < C^*$
- A* expands some nodes with $f(n) = C^*$
- A* expands no nodes with $f(n) > C^*$

### Dominance of admissible heuristics

For admissible heuristics: h2 dominates h1 if $h2(n) \geq h1(n)$ → is better for search, expands less nodes

If h1,h2 admissible: $h(n) = \max(h1(n), h2(n))$ is also admissible and dominates h1 and h2

### Relaxed problems

One can find admissible heuristics by relaxing the exact problem

e.g. 8-puzzle tile can move anywhere or better: to any adjacent tile

Traveling salesperson problem (TSP) -> Relaxation with Minimum spanning tree (MST)

## Local Search

### Iterative improvement

For many problems one does not need a path, but only the goal state →

- State space = set of "complete" configurations
- Find optimal configuration
- Find configuration satisfying constraints

Use iterative improvement algorithms and keep only a single "current" state

E.g. TSP start with complete tour and perform pairwise exchanges,

n-queens move queen to reduce conflicts

### Hill-Climbing (gradient ascent/descent)

Choose neighbouring solution if it is better → finds local maximum in state space landscape

Possibilities: random-restart after a step limit, choose neighbours stochastically or first-choice

### Simulated annealing

Idea: escape local maxima by allowing some bad moves, but gradually decreasing their size/frequency

~ cooling off process of materials

If temperature T is reduced slowly enough, best state x* is reached with probability approaching 1 (can be extremely slow)

### Local beam search

Idea: keep k states instead of 1 → choose top k of all their successors (!= k parallel searches, recruit each other to find good states)

Problem: often all k states end up on same local hill → choose k successors randomly, biased towards good ones

### Genetic algorithms

Genetic algorithm = stochastic local beam search + successors from pairs of states

- Fitness (each state is assigned a fitness value ~quality of state)
- Selection (select a proportion of states)
- Pairs (pair states)
- Cross-Over (create combinations of parent states)
- Mutation (mutate part of state)

Requires states encoded as strings

Crossover helps only if substrings are meaningful components/blocks

### Continuous state spaces

Discretization; e.g. empirical gradient (e.g. of distance from airport) in each coordinate

Gradient methods, solve with iterations..

## Learning from Observations

### Learning

- Modifies agent's decision mechanism to improve performance
- Essential for unknown environments
- As system construction method, instead of writing program in complex setting

Architecture:

- Critic: performance/result assessment

- Performance element: select external actions
- Learning element: make improvements
- Problem generator: suggest actions for new experiences

### Learning element

Makes changes to "knowledge components" (reflex, logical, utility…)

Dictated by: type of performance element, functional component & its representation, available feedback

E.g. taxi

- Performance element: knowledge about driving…
- Critic: assess status, customer reactions…
- Learning element: break softly…
- Problem generator: try brakes on slippery road…

### Representation

- Atomic: states with labels ~ blackbox
- Factored: attributes/values used in planning/constraints
- Structured: dependence model between attributes

### Learning Modes

Feedback:

- Unsupervised: no explicit feedback
- Reinforcement learning: occasional rewards/punishments
- Supervised learning: correct answers for each instance (issue: which examples to choose)

Usually: semi-supervised learning (mix of labelled/unlabelled examples)

Issues: noise, inaccuracies (biased data)

### Inductive Learning (Science)

Simplest form: training set

Given a (finite) training set $(\mathbf{x}_1, y_1),\ldots,(\mathbf{x}_n,y_n)$ of examples, find a function h (hypothesis) that approximates f (target function), f: **x** -> y

If h = f is possible the learning problem is "realizable"

Type of output value to learn:

- Classification: one of finitely many values
- Regression: a (real) number

Ignores prior knowledge and assumes deterministic, observable behaviour, existing examples, that agent wants to learn f.

Construct h to agree with f on training set.

"Consistent" if agreement on all examples

Curve fitting: fit a curve to the training set, e.g. linear, polynomial …

Maximise simplicity under consistency, try not to overfit

Trade-offs between:

- complex hypotheses that fit better vs. simpler hypotheses that might generalize better
- expressiveness of hypothesis space and finding a good hypothesis in it

## Decision Tree Learning

Decision tree: a possible representation for hypotheses. Attribute valued representation

Decision is reached by sequence of tests arranged in a tree

- internal nodes are tests
- child nodes for each possible value
- leaves give decisions

Output understandable (as compared to neural networks)

There are trivial consistent decision trees for any training set but they might not generalize to new examples

Aim: Find a small consistent tree
Idea: Choose "most significant" attribute as root of (sub)tree

Greedy approach, choose next best attribute, try to achieve a flat tree, subsets ideally e.g. "all positive"

Not necessarily good. Better: Information gain

## Information gain

Information in an answer with prior $\langle P_1,\ldots,P_n\rangle$ is its entropy:

$$H(\langle P_1, \ldots, P_n\rangle) = \sum_{i=1}^{n} -P_i log_2 P_i$$

Special case n = 2, $P_2 = 1 - P_1$

$$B(P_1) = H(\langle P_1, P_2\rangle) = -P_1 log_2 P_1 - (1 - P_1) log_2 (1 - P_1)$$

For a set E with p positive and n negative examples and subsets $E_i$ with $p_i$ and $n_i$ and an Attribute A

$$Gain(A) = B\left(\frac{p}{p+n}\right) - Rem(A)$$

Where Rem is the expected number of bits per example over all $E_i$

$$Rem(A) = \sum_i \frac{p_i + n_i}{p + n} B(\frac{p_i}{p_i + n_i})$$

## Overfitting

- consider irrelevant attributes
- break with new examples

- more likely to happen with many attributes
- less likely to happen with more examples

→ Decision tree pruning: using information gain + statistical method -> no irrelevant attributes

→Cross-validation: use part of training data for testing

### Generalizations/Problems

- Missing data(unknown attribute values): weigh possible values with frequency
- Multi-valued attributes(e.g. social security number): normalize Gain(A) with entropy of A
- Continuous, integer valued attributes: use split points (e.g. age > 35)
- Continuous valued output attributes: regression trees, whose leaves are functions of subsets of attributes

### Measuring Learning Performance

- Use theorems of learning theory
- Try h on a new test set

→ Learning curve (% correct on test set per training set size)

Depends on:

- Realizability of target function (missing attributes, restrictive hypothesis space)
- Redundant expressiveness (irrelevant attributes)

## Neural Networks

### Neuron (abstraction)

- Input: linear sum of weighted inputs

$$in_i = \sum_j w_{j,i} a_j$$

  $a_0$ == fixed input "-1" → shift by bias $w_{0,i}$
- Activation function $g(in_i)$
  - Step function (hard limiter)
  - Linear function (threshold function)
  - Sigmoid function $\frac{1}{1+e^{-x}}$ (differentiable: $\frac{d\sigma(x)}{dx} = \sigma(x) * (1 - \sigma(x))$)
- Output $a_i$ <- $g(in_i)$

Can't build arbitrary Boolean functions, e.g. xor → Neural Networks

### Network structure

- Composed of nodes $u_i$ (input, output, processing/hidden)
- Edges/links from $u_i$ to $u_j$ have weight $w_{i,j}$
- f(**x**) mostly depends on edge weights **w** if graph/activation functions are known → set up **w**?

### Feed-forward networks

Layered network: nodes are directly connected only to previous and next layers

- single-layer, multi-layer perceptrons
- no states → reflex agents

## Recurrent networks

- directed cycles with delays (states)
- may be unstable/chaotic
- Hopfield networks
  - All nodes are input & output, bidirectional connections with symmetric weights
- Boltzmann machines
  - State transitions similar to simulated annealing

## Single-Layer Perceptrons

- Output units operate separately, no shared weights → like an independent single perceptron
- Linear separator → can't express XOR (Perceptron with g = step function)

## Multi-Layer Perceptrons

- Hidden units
- Layers (usually) fully connected
- Expressiveness:
  - 2 layers: all continuous functions
  - 3 layers: all functions

## Perceptron learning

Adjust **w** to reduce error on training set (using squared loss and gradient descent)

- Simple update rule $w_j \leftarrow w_j + \alpha \cdot Err \cdot g'\big(in(\boldsymbol{x})\big) \cdot x_j$
- Multi-output learning: split **y** in independent single output learning problems $y_k$

Perceptron vs. Decision Tree

- Perceptron better at majority function
- DTL better at restaurant function (not linearly separable)

## Multi-Layer Perceptron learning

Using additive squared loss $|\mathbf{y} - \mathbf{h_w}(\mathbf{x})|^2$ and gradient descent, errors at hidden units are divided by contributing weights

***Output Layer $a_k$***: same as single-layer perceptron

$$w_{j,k} \leftarrow w_{j,k} + \alpha \cdot a_j \cdot \Delta k \qquad \text{where} \quad \Delta k = Err_k \cdot g'(in_k)$$

***Hidden Layer aj:*** node $a_j$ "responsible" for fraction of errors Δk at output layer

- Back-propagate error from there

$$\Delta j = g'\big(in_j\big) \cdot \sum_i w_{j,k}\, \Delta k$$

- Update rule for weights in hidden layer

$$w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta j$$

[Note: Skipped long derivation of back-propagation loss…]

Problem: find suitable network structure

### Applications

- Prediction of air pollution/electric power prizes/stock developments
- Pattern recognition(handwriting)/face recognition
- …

### Advantages

- Parallelism
- Good for pattern recognition
- Fault tolerant
- Able to learn/improve

### Disadvantages

- Choice of layers/units requires skill
- Needs sufficient training material
- Results/behaviour difficult/impossible to understand
- Implicit knowledge

## Constraint Satisfaction Problems (CSP)

Standard search problem:

- State == blackbox, represented by arbitrary data structure for successor&heuristic function, goal test

CSP:

- States/goal test conform to standard, structured, simple representation
- Use more powerful general-purpose instead of problem-specific heuristic

### Formal Definition

- Finite set V = $\{V_1, V_2, \ldots, V_n\}$ of *variables*
- For each $V_i$ a non-empty *domain* $D_i$ of possible values
- A finite set C = $\{C_1, \ldots, C_m\}$ of constraints (subsets of Cartesian products of values)
  - Unary constraints: {1,2,5,..}
  - Binary constraints: {(1,2),(3,5),…}
  - …

State: assignment of values to some variables.

If assignment does not violate constraints: *consistent*/*legal*

If assignment mentions every variable: *complete*

*Solution*: satisfies all constraints

CSPs might have objective function: *constrained optimization problems*

### Constraint graph

(Hyper)graph that depicts the constraints of a problem. Constraints are (hyper)edges

### Varieties of CSPs

- Discrete CSPs with finite domains. $O(d^n)$ assignments. d.. max domain size, n variables
  - Boolean CSPs
- Discrete CSPs with infinite domains
  - Needs constraint language, can't enumerate all combinations

    - Soluble only for linear constraints on integer values
- Continuous CSPs
    - Linear programming methods

## CSP as standard search problems

Incremental formulation

- Initial state: empty assignment
- Successor function: assign value to unassigned variable according to constraints
- Goal test: assignment is complete

## Backtracking search

Reformulation of CSP:

- Variable assignments are commutative
- Choose only values for a single variable at a node

DFS for CSPs with single-variable assignments: ***backtracking search***

## Improvements to backtracking search

Minimum-remaining-value heuristic MRV:

- Choose variable with fewest (remaining) legal values

Degree heuristic:

- Select variable involved in the largest number of constraints (helps choosing a first variable)

Least-constraining-value heuristic:

- Prefer value that rules out fewest choices for neighbouring variables in constraint graph

Forward checking:

- When variable X is assigned, look at each variable Y that is connected to X by a constraint
  → delete any value from domain of Y that is inconsistent with X

## Arc consistency

Simplest form of constraint propagation (propagate implications of constraint on var onto vars)
"Arc" refers to directed arc in constraint graph

X→Y is *consistent* if for every value of X there is some allowed value of Y

## Further techniques

Intelligent backtracking: go to back to one variable that caused the failure in the conflict set
e.g. backjumping (earliest variable in conflict set)

Local search is effective for CSPs

Take structure of constraint graph into account

# Planning

Come up with a sequence of actions to achieve a goal
e.g. represent actions as first-order logic expressions

## Problems

Frame problem:

- How to represent things that stay unchanged after an action

Ramification problem:

- Representation of implicit effects (e.g. if car moves, driver moves too)

Qualification problem:

- Required preconditions ensuring that an action succeeds ~ correct conceptualisation of things

## Classical planning environments

- Fully observable, deterministic, finite, static(only change when agent acts), discrete(time,…)

## STRIPS (Stanford Research Institute Problem Solver)

States:

- conjunction of positive literals
  Literals must be ground/variable-free and function-free or propositional
  (e.g. Rich ∧ InJail)
- closed-world assumption (not mentioned conditions are false)

Goals:

- partially specified state
- a state s satisfies a goal g if s contains all the atoms in g

Action schemas:

- action name, parameter list  Fly(p, from, to)
- preconditions
  - conjunctions of function-free positive literals (what must be true before action)
  - variable must be contained in parameter list
- effects
  - conjunction of function-free literals (state change)
  - positive literal is true in resulting state, negative (¬P) makes P false
  - variables must be contained in parameter list

Concrete action is applicable if it satisfies preconditions

Strips assumption:

- literals not mentioned in effects remain unchanged (i.e. frame problem solution)

Solution: action sequence that results in goal state

### Action Description Language ADL
Variation of STRIPS

- allows typing e.g. “p : Plane” and “from != to”
- allow negative literals in states, quantified variables, disjunction in goals
- open-world assumption: not mentioned literals are unknown

### Restrictions of STRIPS/ADL:
- ramifications not represented naturally (e.g. dust particles have to be explicit)
- qualification problem not addressed

## Planning with State-space search
### Planning algorithms
Most straightforward: State-space search

- forward state-space search (progression planning): initial state to goal
- backward state-space search (regression planning): goal to initial state
- partial-order planning

### Progression planning
Initial state: initial state of planning problem

*Step cost* typically 1 (only rarely different costs for different actions in STRIPS planners)

In the absence of function symbols the state space is finite

→ Any complete graph search algorithm (A*) yields complete planning algorithm

### Regression planning
Main advantage: consider only *relevant* actions (relevant action: achieves one of the conjuncts of a goal)

Actions that *do not undo* any desired literals are *consistent*

For a given goal G find an action A that is relevant and consistent and build corresponding predecessor

- positive effects of A in G are deleted
- precondition literals of A are added if not already there

Any standard search algorithm can be used to carry out the search

### Partial-order planning
Regression and progression planning have strictly *linear sequences* of actions (no advantage of *problem decomposition ~ independent subsequences*, e.g. doesn’t matter which shoe you put on first)

Planner that can place two actions in a plan without specifying their order → *partial-order* planner

Implemented as search in space of *partial-order plans*

### Partial-order plans (POP)
Contains sets of

- actions

    - taken from set of actions of planning problem
    - empty plan: only start and finish actions
        - start:
            - preconditions: none, effects: initial state
        - end:
            - preconditions: goal, effects: none
- ordering constraints
    - pair of actions **A ≺ B** (A [any time] before B)
    - cycles **A ≺ B, B ≺ A** (*contradiction*) are not allowed
- causal links
    - $\boldsymbol{A \xrightarrow{p} B}$ "A achieves p for B" (e.g. $RightSock \xrightarrow{RightSockOn} RightShoe$)
    - p is true at least from A until B
    - no *conflicts* allowed (action between A and B with effect ¬p)
- open preconditions
    - planners reduce open preconditions without introducing a contradiction

*Solution* is a *consistent* (no cycles, conflicts) plan with *no open preconditions*

Execution: repeatedly choose any possible next action (linearization into a total-order solution)

### POP Algorithm

Initial plan:

- Actions: Start, Finish
- Ordering constraint: Start ≺ Finish
- Causal links: none
- Open preconditions: preconditions of Finish

Successor function picks open precondition *p* on an action *B* and generates successor plan for every possible *consistent* way of choosing an action *A* that achieves *p*.

Add causal link $A \xrightarrow{p} B$ and ordering constraint A ≺ B to the plan.
If A is a new actions add Start ≺ A and A ≺ Finish.

Resolve conflicts between new causal link $A \xrightarrow{p} B$ and every existing action C by adding B ≺ C or C ≺ A.
Add successor states for either or both if plan remains consistent.

Goal test only needs to check that there are no open preconditions (because plan is consistent).

## Decision Theory

Decision-theoretic agent:

- *Continuous* measure of outcome quality (not only good=goal, bad=not goal)
- *Utility theory* + *probability theory* -> make rational decisions based on *beliefs* and *desires*

Decision theory:

- Simplest form: use desirability of *immediate outcomes*
  → environment is *episodic* (vs. sequential)

Nondeterministic, partially observable environments

States: random variables e.g. Result(a) .. outcome states for taking action a

Probability of outcome s', given evidence **e** and executed action a:

P (Result(a) = s' | a, **e**)

## Utility

Agent's preferences: ***Utility function U(s)***: assign a number to a state ~desirability

Expected utility: average utility value of outcomes weighted by their probability

$$EU(a|\boldsymbol{e}) = \sum_{s'} P(Result(a) = s'|a, \boldsymbol{e})U(s')$$

Maximum expected utility (MEU):

- An agent should choose the action that maximises the expected utility

## Preferences

- $A \succ B$: the agent prefers A over B
- $A \sim B$: indifferent between A and B
- $A \succsim B$: prefer or be indifferent

A, B … states

## Lottery

Set of outcomes of an action can be seen as a *lottery*

A lottery L with outcomes $S_1,\ldots,S_n$ (atomic states or lotteries) that occur with probabilities $p_1,\ldots,p_n$:

$L = [p_1, S_1; \ldots; p_n, S_n]$

## Axioms of Utility Theory

Orderability

- $(A \succ B),(B \succ A)$, or $(A \sim B)$

Transitivity

- $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

Continuity

- $A \succ B \succ C \Rightarrow \exists p\ [p,A;1-p,C] \sim B$

Substitutability

- $A \sim B \Rightarrow [p,A;1-p,C] \sim [p,B;1-p,C]$

Monotonicity

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

Decomposability (compress consecutive lotteries into a single one)

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

Agents that violate these axioms behave irrationally

For preferences that satisfy these axioms, there exists a real-valued function U so that

- $U(A) > U(B) \Leftrightarrow A \succ B$
- $U(A) = U(B) \Leftrightarrow A \sim B$

Utility of a lottery:

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

Utility functions are linearly invariant: U'(S) = aU(S) + b  a > 0

Deterministic environments only need preference ranking: value function / ordinal utility function

Preference elicitation: present choice → observer preferences → determine underlying utility function

## Utility scales

Utility of best outcome: $U(S) = u_\top$

Utility of worst outcome: $U(S) = u_\bot$

*Normalized utilities*: $u_\top = 1$, $u_\bot = 0$

## Utility of Money

Usually monotic preference: more money better

***Expected monetary value (EMV):*** $\sum_i p_i M_i$ (probability * money, for each outcome)

But: different preferences for lotteries involving money (sure outcome vs. chance to win more…)

→ Utility is more significant (utility of money ~ logarithm(amount))

## Risks

*Risk-averse*: $U(L) < U(S_{EMV(L)})$  .. $S_{EMV(L)}$ is state of being handed expected money of lottery

*Risk-seeking*: $U(L) > U(S_{EMV(L)})$  *Risk-neutral*: linear utility-vs-money curve

*Certainty equivalent*: the value an agent will accept to take a lottery (e.g. pay 400$ for coin-toss 1000$)

*Insurance premium*: difference between certainty equivalent and EMV (e.g. 100$)

Insurance industry: people are risk averse → insurance premium is positive

## Irrationality

Decision theory: normative theory ~ how agent should act

Descriptive theory: how agent really does act

## Allais Paradox
Certainty effect: people are attracted to certain gains/sure outcomes

- No need to estimate probabilities
- Distrust legitimacy of stated probabilities
- Emotional state: regret of not having taken sure outcome,…

## Ellsberg Paradox
People elect *known probabilities* rather than unknown ones (e.g. 1/3 chance vs. 0 to 2/3 chance)

## Decision Networks (Influence diagrams)
Mechanism for making rational decisions

- Current state, possible actions, resulting state + its utility
- Chance nodes (oval): random variables
- Decision nodes (rectangles): choice of actions
- Utility nodes (diamonds): represents utility function

Evaluation: […]

## Value of Information
Often: not all relevant information is available before making a decision

Information value theory: agents able to choose to acquire information (of any observable chance var)

Valuable because of the potential to affect agent's physical action

***Value of Information***(generally): difference between expected value of best action before and after information is obtained

## Value of Perfect information (VPI)
(If exact evidence can be obtained)

Initial evidence **e**, current best action α: $EU(\alpha|\boldsymbol{e})$

After evidence $E_j = e_j$: $EU(\alpha_{e_j}|\boldsymbol{e}, e_j)$

Value of $E_j$ is currently unknown → average over all possible values $e_{jk}$:

$$VPI_{\boldsymbol{e}}(E_j) = (\sum_k P(E_j = e_{j_k}|\boldsymbol{e})EU(\alpha_{e_j}|\boldsymbol{e}, E_j = e_j)) - EU(\alpha|\boldsymbol{e})$$

- VPI is nonnegative — $VPI_e(E_j) \geq 0$
- VPI is nonadditive — $VPI_e(E_j, E_k) \mathrel{!}= VPI_e(E_j) + VPI_e(E_k)$
- VPI is order independent — $VPI_e(E_j, E_k) = VPI_e(E_k, E_j)$

## Decision-theoretic Expert Systems
Decision analysis: application of decision theory to actual decision problems

- Decision maker: state preference between outcomes
- Decision analyst: enumerate possible outcomes, elicit preferences to determine best action

Creating a decision-theoretic expert system:

- Create causal model
- Simplify to qualitative decision model
- Assign probabilities
- Assign utilities
- Very and refine, evaluate against test-set/gold standard
- Perform sensitivity analysis (decision sensitive to small changes in probability/utility?)

## Philosophical Foundations of AI

Can machines act intelligently like humans, can they have conscious minds?

Possibility of artificial intelligence depends on definition of AI (e.g. "quest for best agent program for given architecture" → possible)

### Weak AI hypothesis

Machines could act ***as if*** they were intelligent

### Strong AI hypothesis

Machines that do so are ***actually*** thinking

### Turing Test

5 minute conversation → let interrogator guess if conversation was with a program or person

Pass: if program fools interrogator 30% of the time

### Argument from Disability

"A machine can never do X" for some property X

- Be kind, beautiful
- Have humor, make mistakes
- Fall in love…

But computers can do things people can't (easily) do, e.g. complex statistical learning algorithms

### Mathematical Objection

There are mathematical questions that are in principle unanswerable by particular formal systems

- Gödel's incompleteness theorem → construct a Gödel sentence G(F)
  - G(F) is a sentence of F, but cannot be proved within F
  - If F is consistent, then G(F) is true

Turing machines as formal systems are limited by the theorem and claimed to be inferior but:

- Computers are finite (not infinite Turing machines) -> not subject to theorem
- Impossible to prove that humans are not subject to theorem

### Argument from Informality (of behavior)

Computers can do no more than follow a set of rules

Human behavior is too complex to imitate (limited logical set of sentences e.g. Dog(x) → Mammal(x) vs. watching a dog)

→*Qualification problem*

### Argument of Consciousness
Claim: Machine that passes the Turing test only *simulates* thinking

Need to feel emotions vs. being able to be intelligent without emotions

### Mind-Body Problem
*Dualist theory*: mind (thinking) and body (physical processes) exist in two separate realms

But: how can mind control body if separate

*Monist theory/physicalism*: mental states are physical states

### Functionalism:
Mental state: intermediate causal condition between output and input

→ Isomorphic mental states possible (level of abstraction under which implementation doesn't matter)

- ➔ Brain replacement experiment
  - Replace all neurons with electronic devices (does consciousness remain?)

### Biological Naturalism
Mental state: high-level emergent feature caused by low-level physical processes in neurons

→ Mental states can't be duplicated. Would require same causal power between neurons

- ➔ Chinese room

### Ethics and Risks of AI
- People might lose jobs to automation, or have too much/too little leisure time
- lose sense of being unique
- AI might be used towards undesirable ends
- Loss of accountability, end of human race,…