Sequential Decision Making and Reinforcement Learning

Fabio G. Cozman - Office MS08 -
fgcozman@usp.br

November 23, 2021

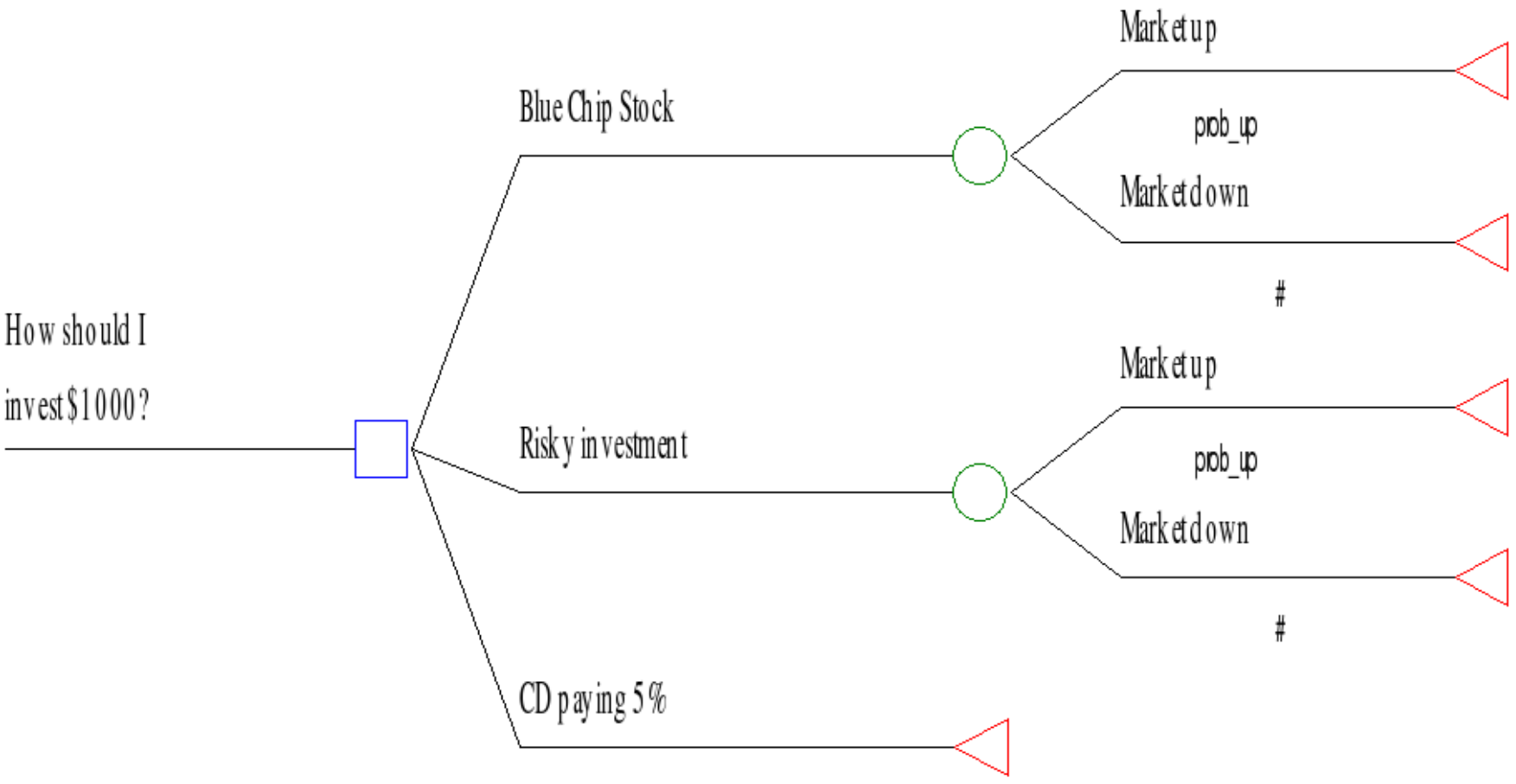
Sequential decision making

- Often we have a single decision to make.
- More often, a sequence of decisions.
- Several models:
  - Decision trees.
  - Influence diagrams.
  - Markov decision processes.
  - Partially observable Markov decision processes.

Decision trees

- A tree with three kinds of nodes:
  - Decision nodes.
  - Chance nodes.
  - Value nodes (always in the leaves).
- Decision and chance nodes are usually interleaved.
- Best sequence of decisions obtained by backward induction.

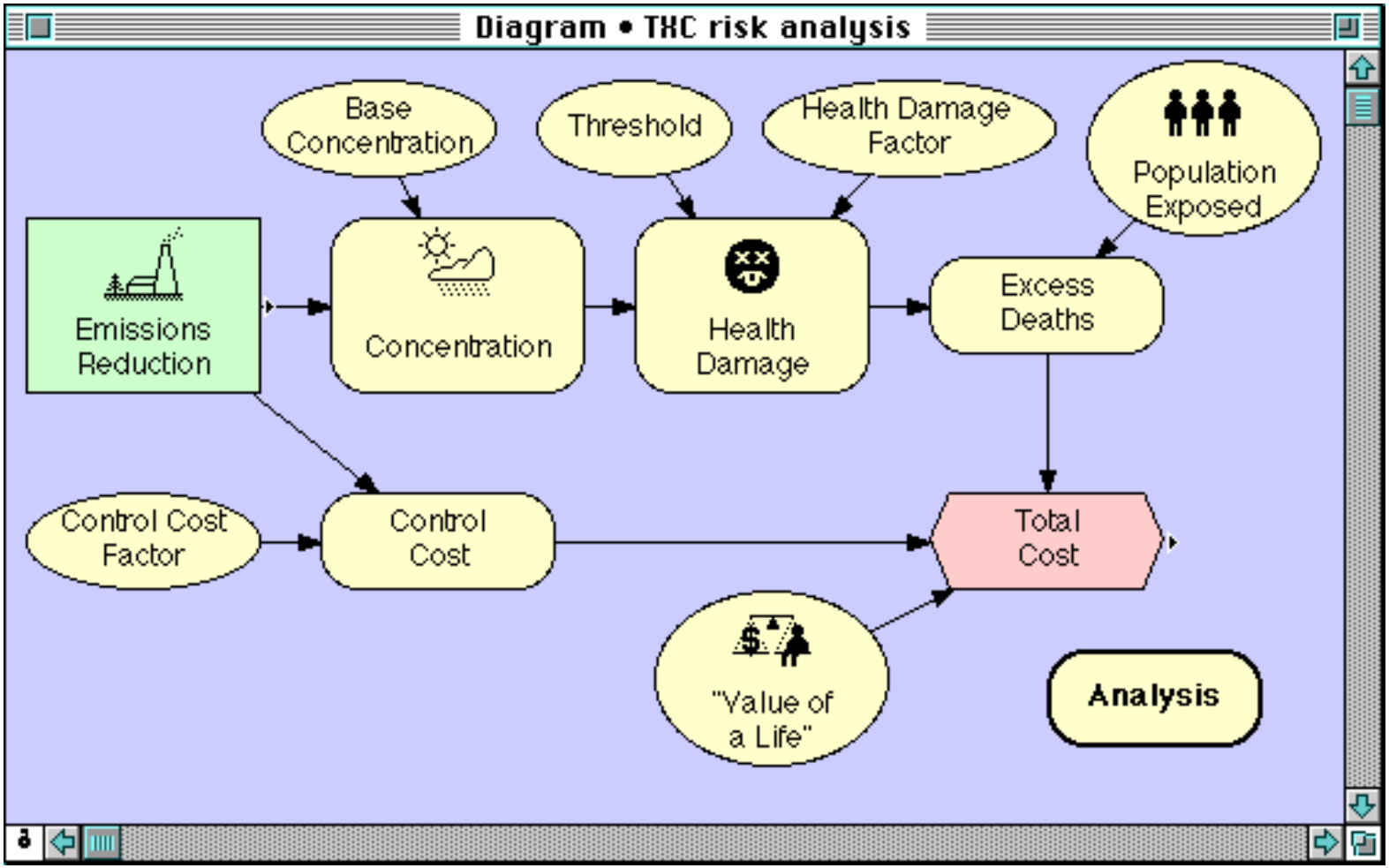
A simple decision tree in TreeAge



Another "language": Influence diagrams

- Directed acyclic graph with chance nodes, decision nodes, and value nodes.
  - Value nodes are assumed as additive.
  - Decision nodes are assumed to recall previous decisions.
- An influence diagram can be "opened" into a (symmetric) decision tree.
- Solution by backward induction (often using Bayesian networks as intermediate representation).

Example: a simple influence diagram in



Markov decision processes (MDPs)

- MDPs are quite popular in economics, management and operations research.
- An MDP consists of
  1. A state space S.
  2. An action space A.
  3. Transition probabilities $p_a(r|s) = P_a(s_{t+1} = r|s_t = s)$.
  4. Costs $c_a(s)$.
- Often represented as graphs where nodes are states.
- Another representation: a transition matrix P_a for each action a.

Policies and their costs

- A *policy* specifies an action for each state (possibly indexed by t).
- A *stationary policy* is a policy that does not depend on t.
- A policy π_1 dominates policy π_2 if π_1 has total cost smaller than π_2.
- But how to measure "cost" of a policy?

Costs

- **Additive cost:** just add costs for all transitions.
- **Discounted cost:** add costs, but with discount γ:

$$c(s_0) + \gamma c(s_1) + \gamma^2 c(s_2) + \dots$$

- **Average cost:** add costs, divide by number of transitions.
- **Goal state:** all costs are ignored, what matters is to reach some state.

Discounted cost

- The most popular, and easiest to handle, is discounted cost.
- We must find the optimal policy π^*:

$$\pi^* = \arg\min_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t c_{\pi(s_t)}(s_t)\right].$$

- For discounted cost, the optimal policy always exists (not necessarily true for other costs!).

Basic relation about discounted cost

- Denote by $E[\pi|s]$ the expected cost when the state is s at $t = 0$.
- Then:

$$E[\pi|s] = c_{\pi(s)}(s) + \gamma \sum_{r \in S} p_{\pi(s)}(r|s) E[\pi|r] \,.$$

- How about the optimal policy and the optimal expected cost?

Bellman equation

- Denote by $E^*[s]$
  - the optimal expected cost when the state is s at $t = 0$;
  - called the *value function* (it depends only on s!).
- By dynamic programming we obtain:

$$E^*[s] = \min_{a \in A} \left(c_a(s) + \gamma \sum_{r \in S} p_a(r|s) E^*[r] \right).$$

- From the optimal cost, we obtain:

$$\pi^*(s) = \arg\min_{a \in A} \left(c_a(s) + \gamma \sum_{r \in S} p_a(r|s) E^*[r] \right).$$

Algorithms

1. Linear programming solution: polynomial algorithm, but rarely used.
2. Value iteration.
3. Policy iteration.

...and many variants of these.

Value iteration

- Start with some function $E_0[s]$ for all $s \in S$ (may even be equal to zero!).
- Now repeat untill convergence:
  For each $s \in S$,

$$E_{i+1}[s] = \min_{a \in A} \left(c_a(s) + \gamma \sum_{r \in S} p_a(r|s) E_i[r] \right) .$$

- Take, from the last iteration:

$$\pi^*(s) = \arg \min_{a \in A} E_N[s] .$$

Convergence of value iteration

- It always converges to the unique optimal policy.
- Convergence is exponentially fast:

$$||E_{i+1}[s] - E^*[s]|| \leq \gamma ||E_i[s] - E^*[s]||$$

(where $||f(x)|| = \max_x |f(x)|$).

- If

$$||E_{i+1}[s] - E_i[s]|| \leq \epsilon(1-\gamma)/\gamma,$$

then

$$||E_{i+1}[s] - E^*[s]|| \leq \epsilon;$$

thus we know when to stop the iterations.

Policy iteration

- Start with some policy π_0.
- Repeat:
  1. Solve (note that this is a linear system):

$$E[\pi_i|s] = c_{\pi_i(s)}(s) + \gamma \sum_{r \in S} p_{\pi_i(s)}(r|s) E[\pi_i|r] \,.$$

  2. Find $a \in A$ such that, for some $s \in S$,

$$c_a(s) + \gamma \sum_{r \in S} p_a(r|s) E_i[r] \leq E_i[s] \,.$$

     - If there is such a, then make $\pi_{i+1}(s) = a$.
     - Otherwise, stop policy iteration.

Convergence of policy iteration

- It always converges to the unique optimal policy.
- Speed of convergence is not known, but empirically observed to be quite fast.

Factored representations

- Usually MDPs represent states explicitly.
- However, representations in terms of variables are more compact.
- Factored representations use Bayesian networks to represent $P_a(r|s)$ (a dynamic Bayesian network indexed by actions).
- There are graphical representations for costs and policies as well.

Planning in AI

- One of the oldest and central themes in AI is planning:
  - Given a (logical) description of a problem,
  - find a sequence of actions to get to a goal.
- This is sequential decision making!
- Classical planning is just deterministic decision making; probabilistic planning is usually equivalent to MDPs.

Planning languages

- Distinguishing feature of planning (and AI): representation is important.
- There has been significant research on representation languages (STRIPS, PDDL, PPDDL...).
- PPDDL:
  (:action buy-coffee
  :effect
  (and (when (not (in-office) (probabilistic 0.8
  (has-coffee))))))
- RDDL: language that specifies planning problems using factored MDPs.

RDDL

- Relational Dynamic Influence Diagram Language.
- Language of the 2011 Planning Competition.
- Basic idea: DBN+value+decisions with parameters...
- Everything is a parameterized variable, couple with logical/arithmetic/conditional expressions
- Basic probability distributions in the language.
- Plus concurrency, constraints, rewards.

RDDL example

```
pvariables {
 p : {state-fluent, bool, default = false};
 r : {state-fluent, bool, default = false};
 a : {action-fluent, bool, default = false};
};
cpfs {
 p' = if (p ^ r)
      then Bernoulli(0.9)
      else Bernoulli (0.3);
 r' = if (~q) then KronDelta(r)
              else KronDelta (r <=> q);
};
reward = p + q - r;
```

More on RDDL

- Possible to use enum and int and real variables.
- Possible to define intermediate and observable variables.
- Possible to define concurrency and constraints.
- Possible to use summations and products:

```
reward =
    sum_{?x : x_pos, ?y : y_pos}
        alive(?x, ?y);
```

- ...and MANY other features...

Reinforcement learning

- Basic idea: learning to behave based only on reward/punishment after actions.
- Often thought of as the estimation of an MDP based only on information about the cost of each visited state.
  - Sometimes learning is passive, sometimes active (that is, with active exploration).
- Often it is not necessary to learn the whole MDP; just to search for the optimal policy.
- Often just the value function is learned; sometimes just approximations of the value function.

Q values

- $Q(s, a)$ is the expected discounted future reward for starting in state s, taking a as the first action, and then continuing optimally.
- So:

$$\pi^* = \arg\max_a Q^*(s, a).$$

- The agent only needs to consider each available action a in its current state s and chooses the action that maximizes $Q(s, a)$.
- $Q(s, a)$ summarizes all the information needed to determine the discounted cumulative reward that will be gained in the future if a is selected in s.

Q-learning

Idea: estimate the Q^* function directly, without estimating transition probabilities.

1. Set learning rate α, initialize $Q(s, a)$ arbitrarily.
2. Observe the current state s_t
3. Do forever:
   1. select an action a_t and execute it in s_t;
   2. receive immediate reward $r(s_t, a_t)$;
   3. observe the new state s_{t+1};
   4. update $Q_{t+1}(s, a) \leftarrow$

$$(1-\alpha)Q_t(s_t, a_t)+\alpha[r(s_t, a_t)+\gamma \max_a Q_t(s_t+1, a)]$$

   5. $s_t \leftarrow s_{t+1}$

Learning rate

- The basic form of the update looks like this:

$$X_{t+1} \leftarrow (1 - \alpha)X_t + \alpha New_t,$$

  where α is the learning rate, usually about 0.1 or 0.2.
- Essentially a running average of the new terms received in each step.
- With a small α, the system will converge slowly but the estimates will not fluctuate very much.
- It is typical (and, in fact, required for convergence), to start with a large α, and then decrease it over time.

Two ways to look at this:

- One is the usual kind of averaging we do when we collect many samples and try to estimate their mean (using the learning rate).
- The other is the dynamic programming iteration done by value iteration, updating the value of a state based on the estimated values of its successors.

Convergence guarantees

- Result: The optimal Q function is produced if the world is really an MDP, if we manage the learning rate correctly, and if we explore the world in such a way that we never completely ignore some actions and states.

Challenges in Q-learning

- Large or continuous state spaces: usually require function approximators (for instance, neural networks, regression trees, etc) that lose convergence guarantees but that work well.
- Slow convergence: many applications build a simulator and use it to generate many samples that allow the system to learn to behave first in a controlled environment.

Exploration and exploitation

- Learning to decide sequentially requires one to choose between:
  - Exploitation: Make the best decision given current information.
  - Exploration: Gather more information and decide later.
- There are many algorithms in Reinforcement Learning that deal with this trade-off.