Quantitative Decision Making in Business

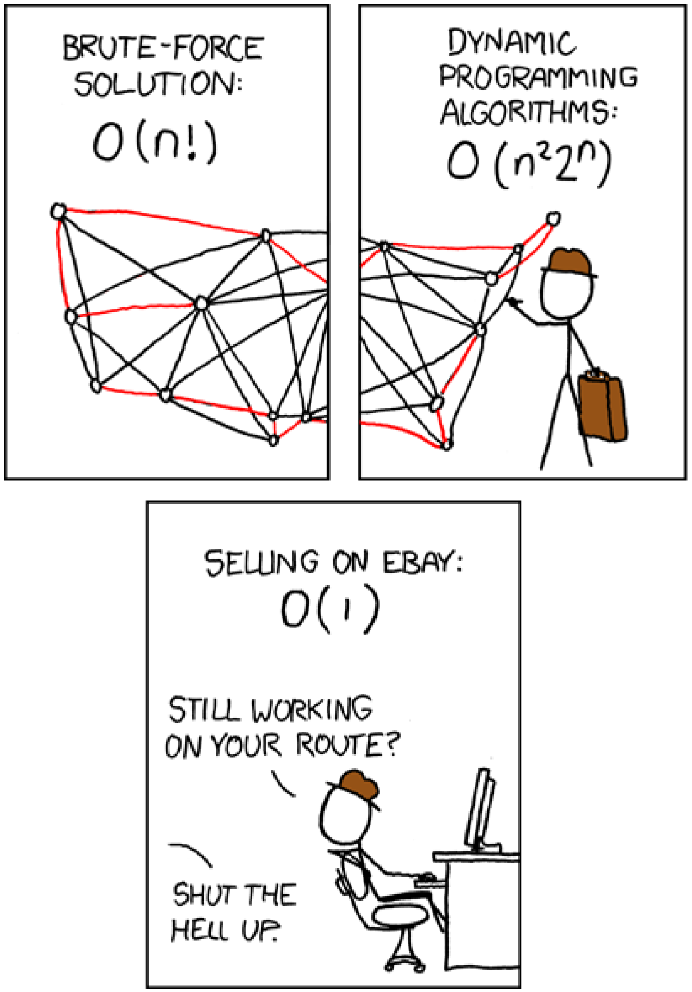
Summer University 2018

Topic 11: Basics of Dynamic Programming

Technische Universität Berlin

Lecturer:
Lissy Langer, M.Sc.
Chair of Production and Operations Management

2. Basics of Dynamic Programming



Picture https://xkcd.com/399/ (The travelling salesman problem)

a. Introduction
   - Games to play
   - A Machine Replacement Model
   - Predictive Maintenance

b. Dynamic Programming Basics
   - Components of a finite-horizon Markov Decision Process
   - The Optimality Equations

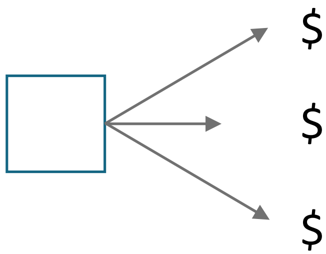
c. Solving the Optimality Equations

Case Study 3: Sea Crest B&B

What is new…

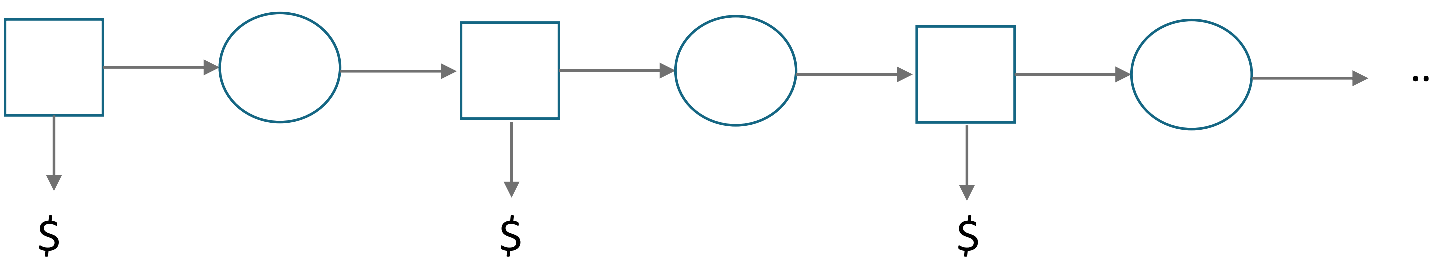
so far:

one-shot or ***episodic decision problems*** *in which the utility (or expected value) of each action is known*



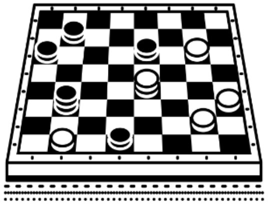
from now on:

sequential decision problems *in which the agent's utility depends on a sequence of decisions. The best action is not always the greedy one.*



Russell, Norvig (2010)

there are some games to play…

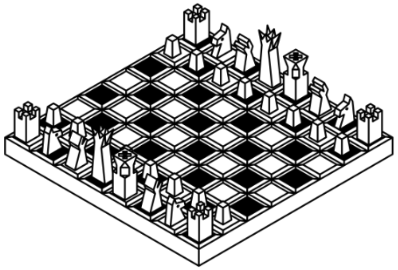


10x10
$500*10^{15}$

CHECKERS: in 2007 the game was weakly solved by the help of CHINOOK: From the standard starting position, both players can guarantee a draw with perfect play. It took 18 years to "learn".

"In checkers, the number of possible moves in any given situation is so small that we can confidently expect a complete digital computer solution to the problem of optimal play in this game." (Bellman, 1965)

Though actually there are around 500 quadrillion possible positions which is 500,000,000,000,000,000 or $500*10^{15}$.



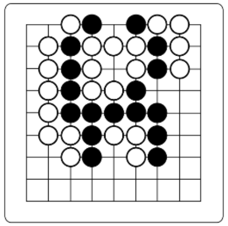
8x8
10^{43}

CHESS: grand prize of $100,000 was awarded to DEEP BLUE in 1997

"The decisive game of the match was Game 2, which left a scar in my memory . . . we saw something that went well beyond our wildest expectations of how well a computer would be able to foresee the long-term positional consequences of its decisions. The machine refused to move to a position that had a decisive short-term advantage—showing a very human sense of danger." (Kasparov, 1997)

Russell, Norvig (2010)

there are some games to play…



19x19
$2x10^{170}$

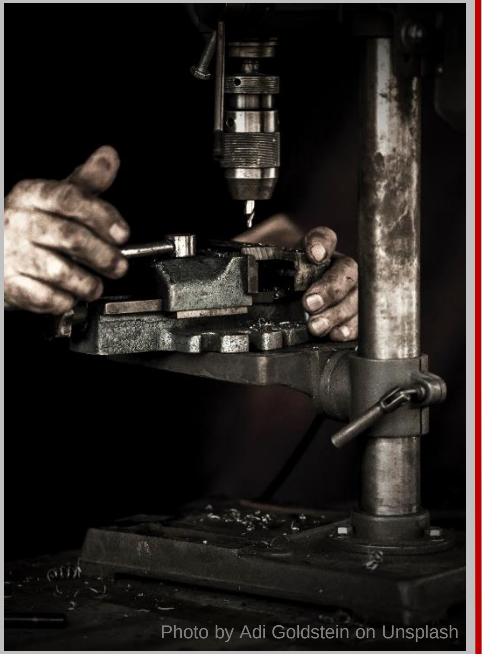
GO: "you may place your stone (playing piece) on any point on the board, but if I surround that stone, I may remove it."

Now the best programs play *most* of their moves at the master level; the only problem is that over the course of a game they usually make at least one serious blunder that allows a strong opponent to win.

Russell, Norvig (2010)

and Atari games…

https://www.youtube.com/watch?v=V1eYniJ0Rnk&vl=en



Example 1: Machine Replacement

Consider a machine that has to be operated throughout a planning horizon of N periods. The decision maker decides whether to replace the machine at the beginning of every year or not.

The problem of interest is to determine a cost-minimizing replacement policy under the following set of assumptions:

- The annual operating cost of an i-year-old machine is **c(i)**;
- The price of a new machine is **b**;
- At the end of the planning horizon, an i-year-old machine can be sold at salvage value **s(i)**.
- You face a discount factor of **α**.

Example:
Suppose N = 3, starting age = 2, b=65€, α=1.

	i=0	i=1	i=2	i=3	i=4	i=5
c(i)	10	20	33	50	70	
s(i)		30	15	10	5	0

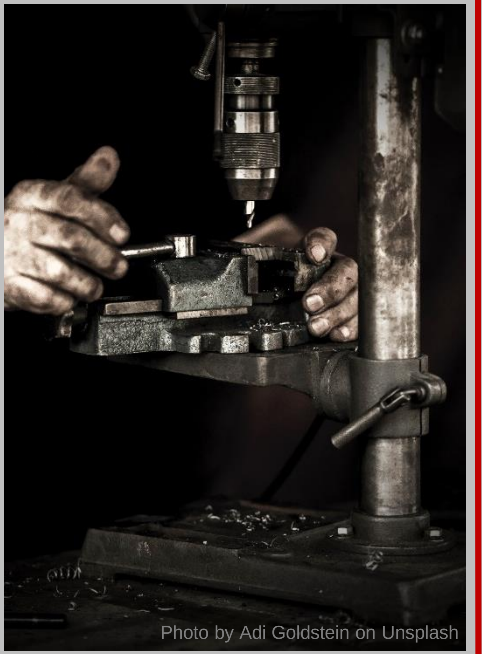


Photo by Adi Goldstein on Unsplash

Example 1: Machine Replacement

n	x_n	Replace (R)	Don't Replace (D)	Max E[Reward] in state x at stage n	Best action

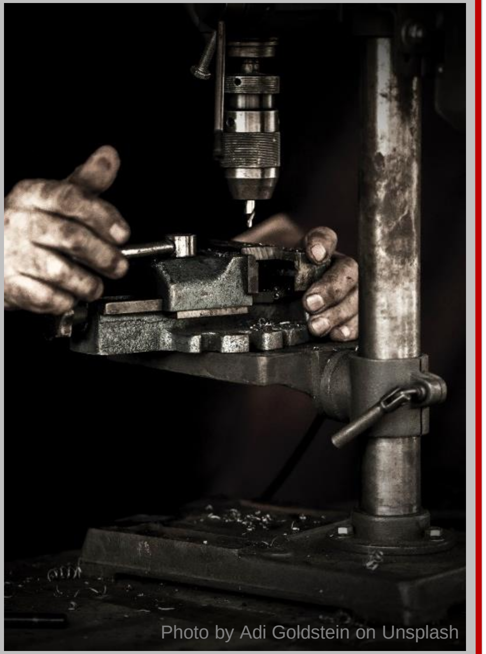


Photo by Adi Goldstein on Unsplash

Example 2: Predictive Maintenance

Consider a machine that has to be operated throughout a planning horizon of N periods. The decision maker decides whether to replace the machine at the beginning of every year or not.
The problem of interest is to determine a cost-minimizing replacement policy under the following set of assumptions:

- The annual operating cost of an i-year-old machine is c(i);
- The price of a new machine is b;
- At the end of the planning horizon, an i-year-old machine can be sold at salvage value s(i).
- You face a discount factor of α,
- **the machine faces a certain probabilty p(i) of breaking down (and then has to be replaced at the end of the period with additional costs c).**

Example:
Suppose N = 3, starting age = 2, b=65€, α=1, c=35€.

	i=0	i=1	i=2	i=3	i=4	i=5
c(i)	10	20	33	50	70	
s(i)		30	15	10	5	0
p(i)	0	0.05	0.1	0.2	0.35	0.4

Example 2: Predictive Maintenance

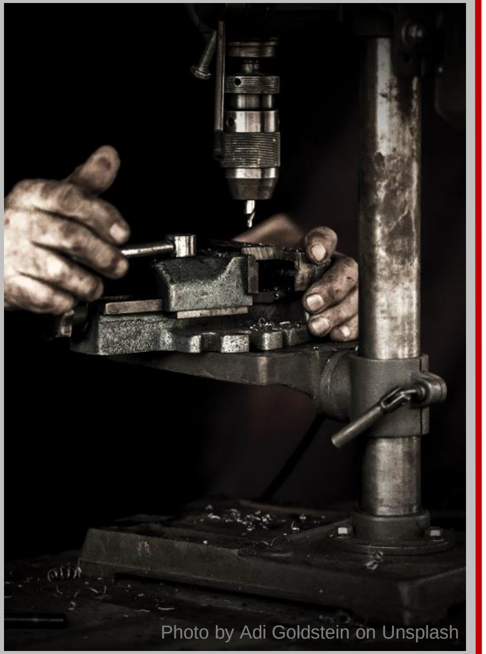


Photo by Adi Goldstein on Unsplash

n	x_n	Replace (R)	Don‘t Replace (D)	Max E[Reward] in state x at stage n	Best action

Sequence of events

$$X_0 \xrightarrow{a_0} X_1 \xrightarrow{a_1} X_2 \xrightarrow{a_2} X_3$$

Total rewards:

$$r_0(x_0,a_0) \quad + \quad \alpha\, r_1(x_1,a_1) \quad + \quad \alpha^2 r_2(x_2,a_2) \quad + \quad \alpha^3 r_3(x_3,a_3)$$

Maximize expected discounted rewards:

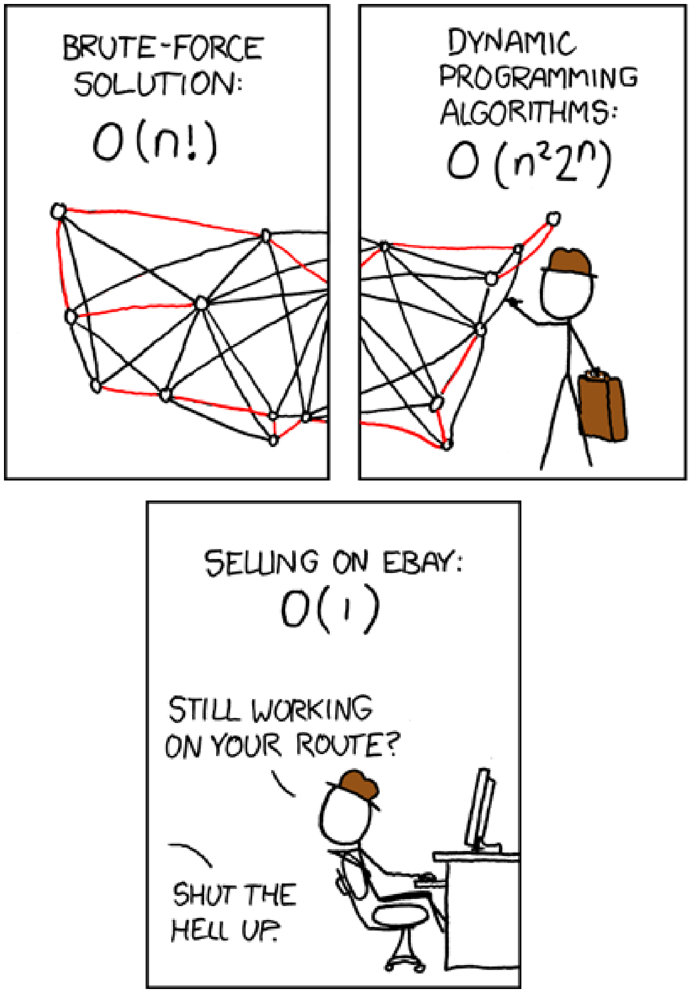
$$E[\ r_0(x_0,a_0) \quad + \quad \alpha\, r_1(x_1,a_1) \quad + \quad \alpha^2 r_2(x_2,a_2) \quad + \quad \alpha^3 r_3(x_3,a_3)\ ]$$

Task is to find a policy π so that whenever we are in state x at time n we will perform action a. This policy will generate the following value:

$$V^\pi(x_n) = r(s_n, a_n) + \alpha \sum_{x'_{n+1} \in \mathbb{X}_n} p_n(x_n, a_n, x'_{n+1}) V^\pi(x'_{n+1})$$

The optimal policy is given by: $V^*(x_n) = \max_\pi V^\pi(x_n)$

2. Basics of Dynamic Programming



Picture https://xkcd.com/399/ (The travelling salesman problem)

a. Introduction
   - Games to play
   - A Machine Replacement Model
   - Predictive Maintenance

b. Dynamic Programming Basics
   - Components of a Stochastic Dynamic Program (Markov Decision Process)
   - The Optimality Equations

c. Solving the Optimality Equations

Case Study 3: Sea Crest B&B

Technische Universität Berlin
Lissy Langer, M.Sc.
Chair of Production and Operations Management

Markov assumption (memorylessness)

Andrei Markov
(1856-1922)

The current state only depends on a *finite fixed number* of previous states.

In the **first-order Markov process**, the current state depends only on the previous state and not on any earlier states. In other words, a state provides enough information to make the future conditionally independent of the past, and we have

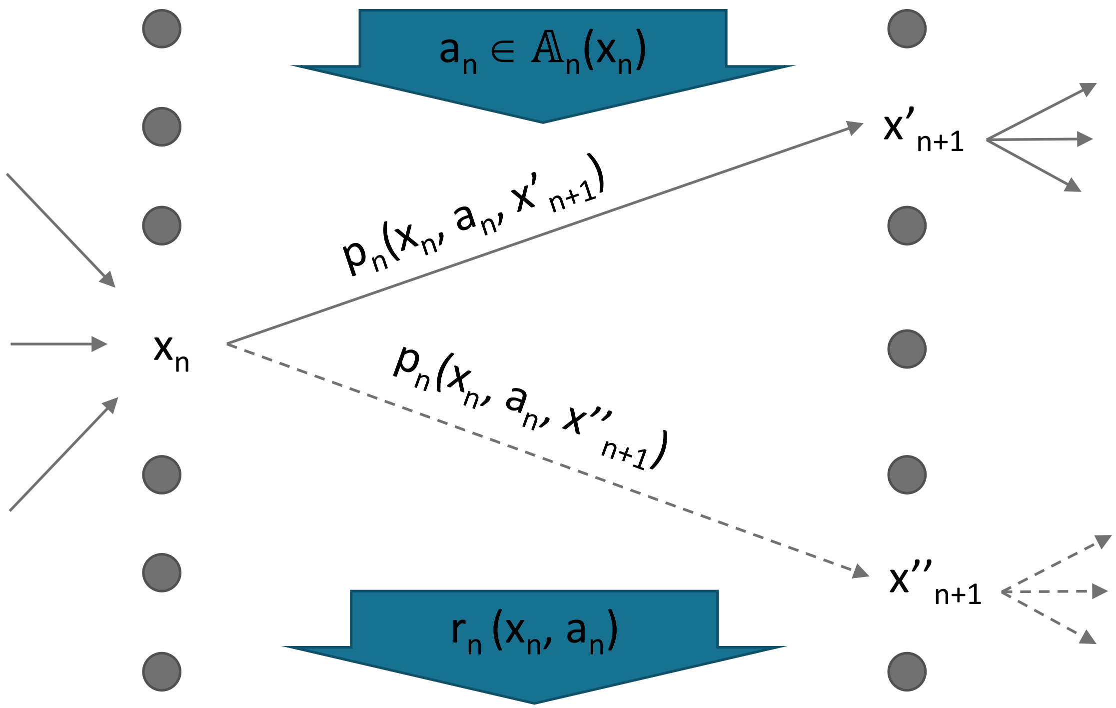
$$\mathrm{P}(X_{t+1} = x' | X_t = x_{t,} A_t = a_{t,} X_{t-1} = x_{t-1,} A_{t-1} = a_{t-1,\dots,} X_0 = x_0) = \mathrm{P}(X_{t+1} = x' | X_t = x_{t,} A_t = a_t)$$

So, a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a **Markov decision process**, or **MDP**, and consists of a set of states (with an initial state x_0); a set $\mathbb{A}(x)$ of actions in each state; a transition model P(x'| x, a); and a reward function r(x). For finite-horizon problems these also depend on the time n.

Russell, Norvig (2010)

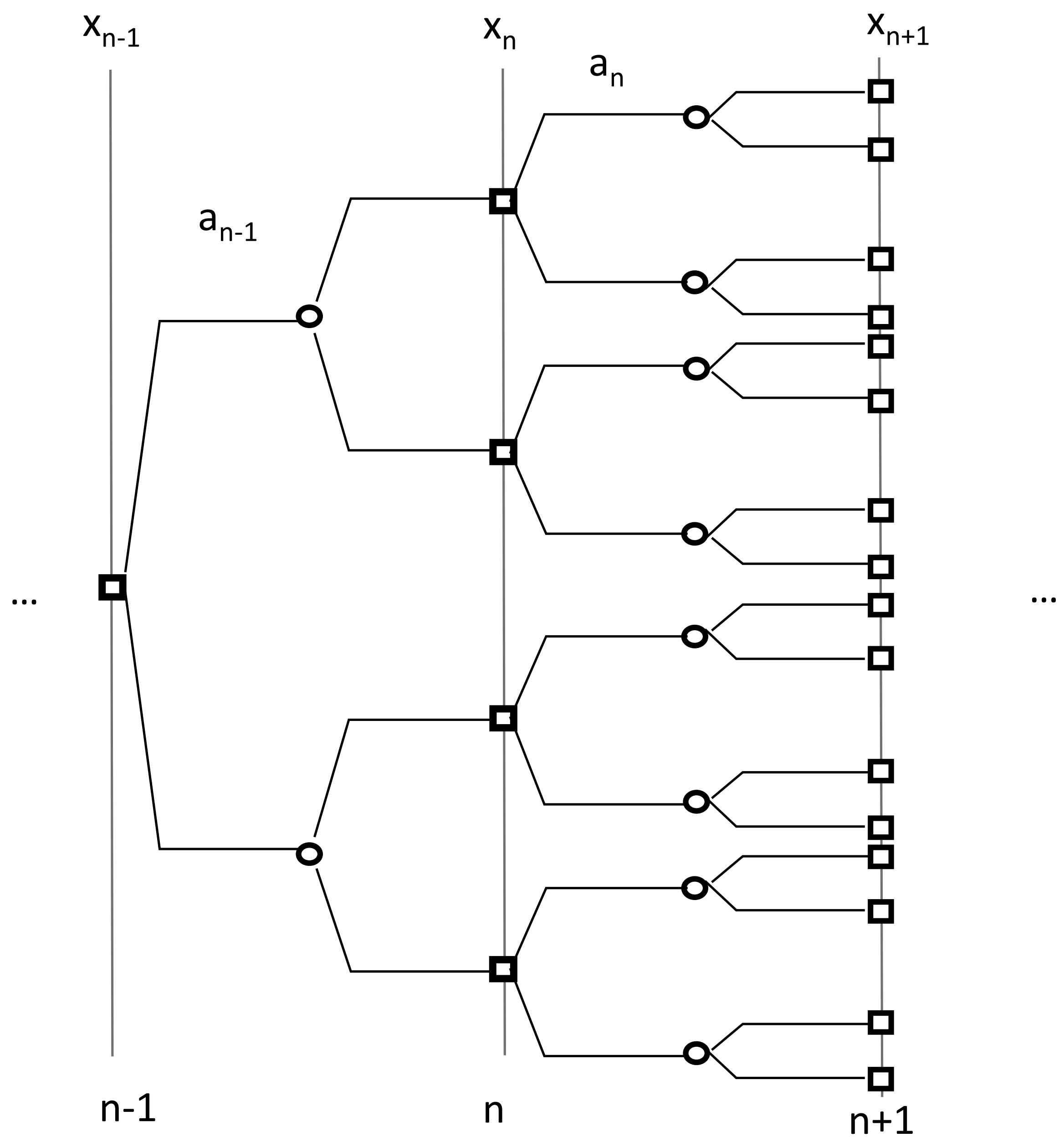
How to think about a finite-horizon Markov decision process…

A finite-horizon Markov Decision Process (MDP) describes a stochastic system that is observed at discrete times n = 0,…,N. If at time n system state x_n from the state space $\mathbb{X}$ is observed, a decision-maker chooses an action a_n among the admissible actions $\mathbb{A}_n(x_n)$. This action results in an immediate one-stage reward $r_n(x_n,a_n)$ and a transition to system state x_{n+1} at time n+1 with probability $p_n(x_n,a_n,x_{n+1})$. At time n = N a terminal reward $V_N(x_N)$ is gained and the sequence is stopped.



How to think about a finite-horizon Markov decision process…

for finite state and action spaces, we know $|X| < \infty$ and $|A| < \infty$



Dynamic Programming Basics

Components
$(N, \mathbb{X}, \mathbb{A}, p, r, V_N, \alpha)$

Components of a finite-horizon MDP

(i) The **planning horizon** $N \in \mathbb{N}$;

(ii) The countable **state space** $\mathbb{X}$ with $\mathbb{X}_n \subseteq \mathbb{X}$ denoting the non-empty subset of possible states in period n=0,..,N;

(iii) The countable **action spaces** $\mathbb{A}_n$. $\mathbb{A}_n(x)$ is the non-empty finite set of admissible actions in state $x \in \mathbb{X}_n$ at time $0 \le n < N$; the union of all n-stage action spaces is $\mathbb{A}$;

(iv) Transition laws p_n: $\mathbb{K}_n = \{(x, a) | x \in \mathbb{X}_n, a \in \mathbb{A}_n(x)\} \times \mathbb{X}_{n+1} \rightarrow [0,1]$, which represent the probability $p_n(x,a,x')$ for a **transition** from state $x \in \mathbb{X}_n$ to $x' \in \mathbb{X}_{n+1}$ given action $a \in \mathbb{A}_n(x)$ at time $0 \le n < N$;

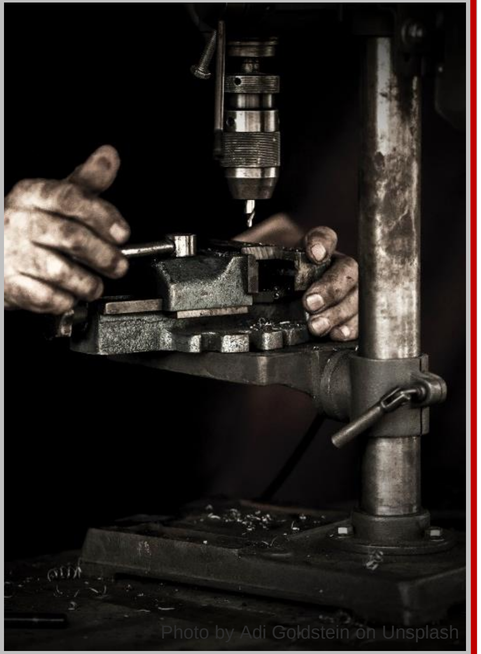
(v) Immediate **reward** functions r_n: $\mathbb{K}_n \rightarrow \mathbb{R}$, which represent the reward $r_n(x, a)$ for choosing action a in state x at time $0 \le n < N$;

(vi) **Terminal reward** V_N: $\mathbb{X}_N \rightarrow \mathbb{R}$, which represents the reward $V_N(x)$ for ending in state x at time N;

(vii) One-stage **discount factor** $0 \le \alpha \le 1$.

Dynamic Programming Basics

Components
$(N, \mathbb{X}, \mathbb{A}, p, r, V_N, \alpha)$



In Example 1: Machine Replacement

(i) N

(ii) $\mathbb{X}_n$

(iii) $\mathbb{A}_n$

(iv) p_n

(v) r_n

(vi) V_N

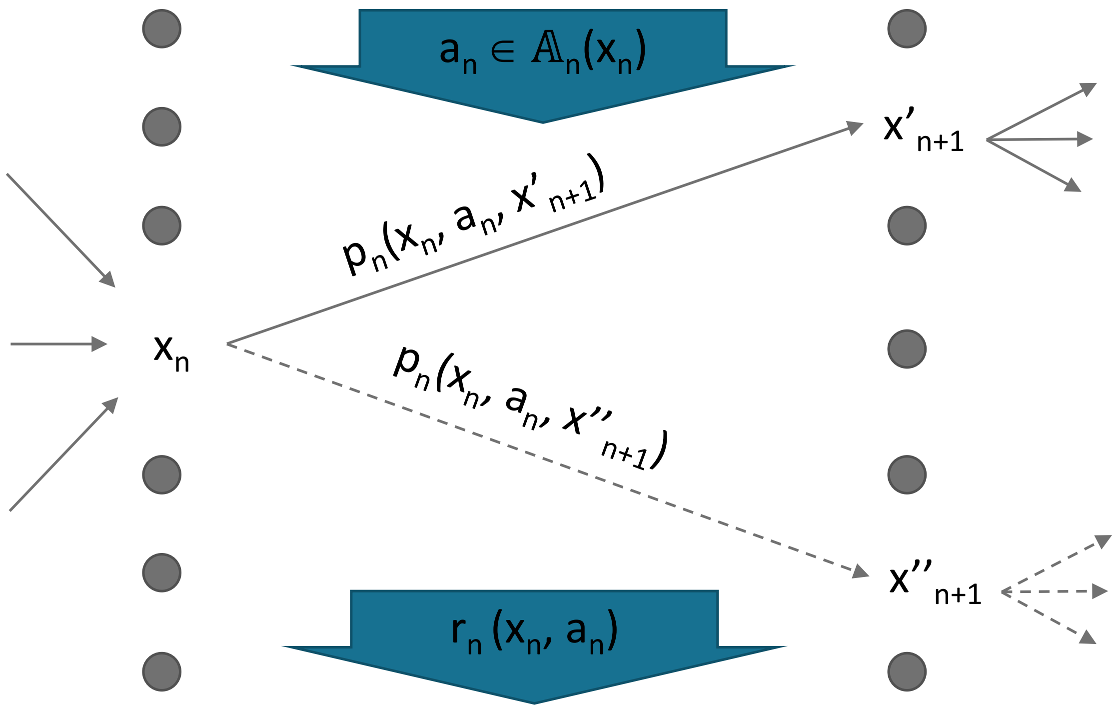
(vii) α

Solution approach

A solution must specify what the agent should do for *any* state that the agent might reach.

A solution of this kind is called a **policy**. It is traditional to denote a policy by π, and π(x) is the action recommended by the policy π for state x.

If the agent has a complete policy, then no matter what the outcome of any action, the agent will always know what to do next.



Components
$(N, \mathbb{X}, \mathbb{A}, p, r, V_N, \alpha)$

Decision rule $f_n(x)$
Policy π
Value function $V_n(x)$

Definitions

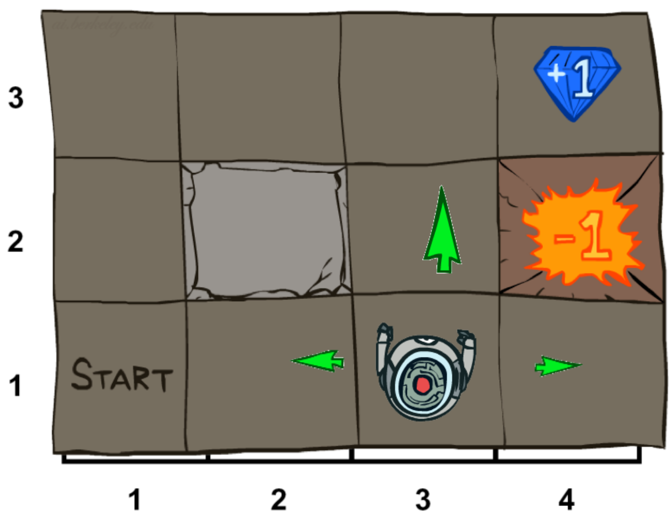
- Let f_n:X $\rightarrow \mathbb{A}_n(x)$ be a deterministic Markovian **decision rule**, which specifies the action $f_n(x) \in \mathbb{A}_n(x)$ to be taken in state x at time $0 \le n < N$, $f_n \in \mathbb{F}_n$, the set of all deterministic Markovian decision rules;
- Let $\pi = (f_0, \ldots, f_{N-1}) \in \Pi = \mathbb{F}_0 \times \mathbb{F}_1 \times \ldots \times \mathbb{F}_{N-1}$ be an N-stage deterministic Markovian **policy**;
- The **expected total discounted reward** under policy π over time periods n,n+1,..,N is random, it is
  $R^{\pi}_{N,\alpha}(x) = E^{\pi}_{x}\{\sum_{n=0}^{N-1}[\alpha^n r_n(X_n, f_n(X_n))] \; + \alpha^N V_N(X_N)\}$;
- **Goal**: find the policy that maximizes $R^{\pi}_{N,\alpha}(x)$
- **A policy π* is called optimal** if
  $R^{\pi *}_{N,\alpha}(x) \ge R^{\pi}_{N,\alpha}(x)$ for all $x \in \mathbb{X}$, $\pi \in \Pi$;
- **The value function** is defined as the maximum expected reward that can be achieved starting from state x at time n
  $$V_n(x) = \sup_{\pi \in \Pi} R^{\pi}_{N,\alpha}(x);$$

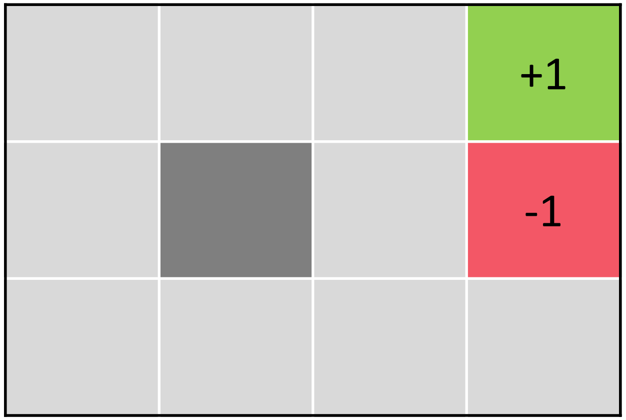
Grid world -deterministic

$r = -0.04$, $\alpha=1$

$V_T(x)=0$

deterministic movements

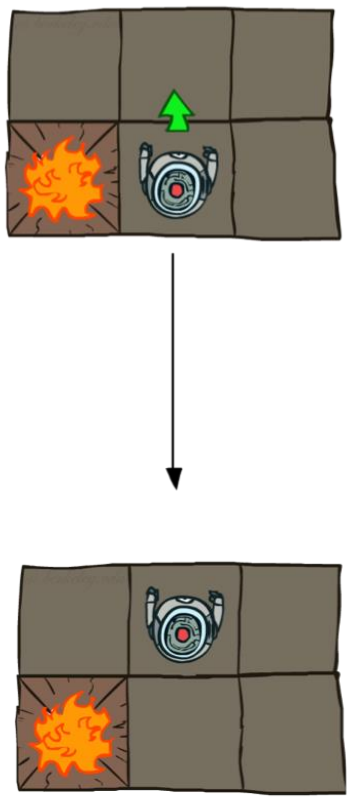




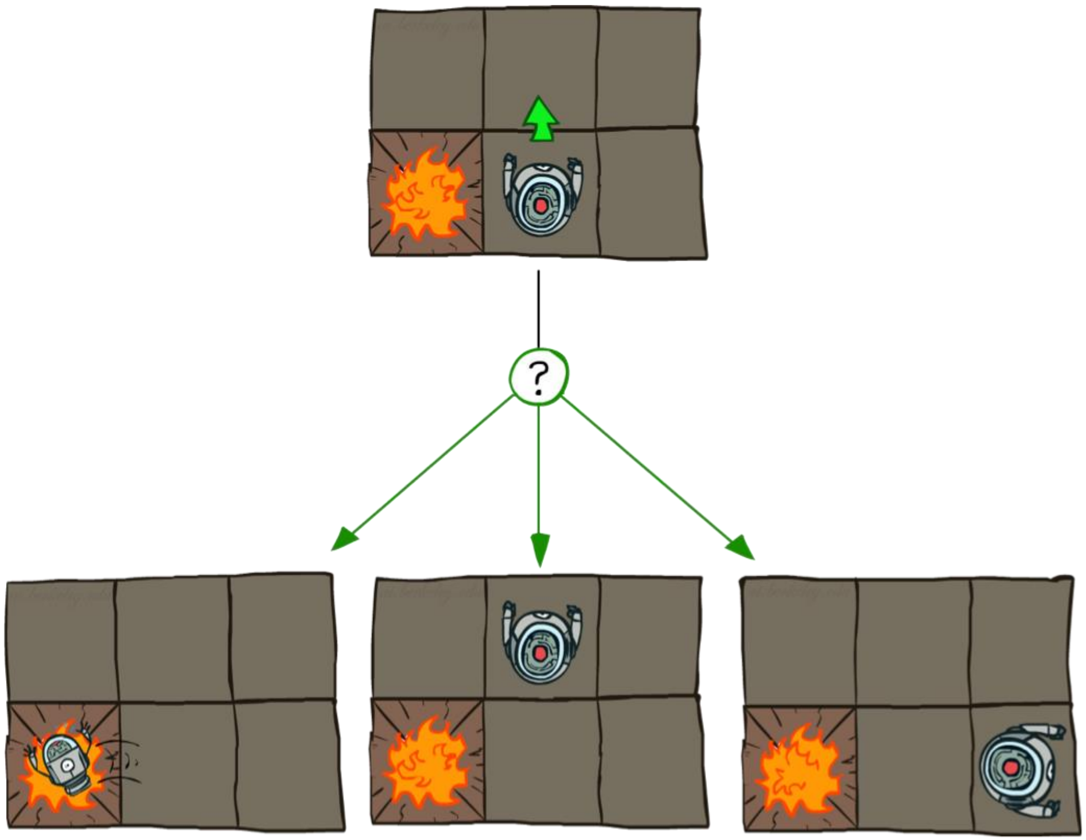
Klein, Abbeel, CS 188, UC Berkeley

Grid world - stochastic

Deterministic Grid World



Stochastic Grid World



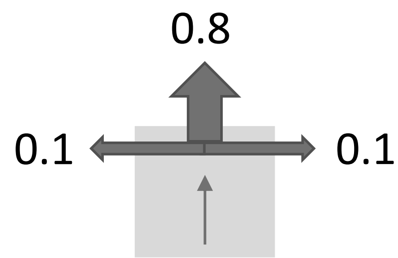
Klein, Abbeel, CS 188, UC Berkeley

Grid world – finite-horizon

r = -0.04, α=1

$V_T(x)=0$

stochastic movements



N=1

·	·	→	+1
·		←	-1
·	·	·	↓

N=3

→	→	→	+1
·		↑	-1
·	·	↑	↓

N=100

→	→	→	+1
↑		↑	-1
↑	←	←	←

→ in a finite-horizon environment we have a non-stationary optimal policy

Grid world – differing rewards

$V_T(x)=0$, $\alpha=1$
stochastic movements

r(x) = -2

→	→	→	+1
↑		→	-1
→	→	→	↑

r(x) = - 0.4

→	→	→	+1
↑		↑	-1
↑	→	↑	←

r(x) = -0.01

→	→	→	+1
↑		←	-1
↑	←	←	↓

0 < r(x)

·	·	←	+1
·		←	-1
·	·	·	↓

Technische Universität Berlin
Lissy Langer, M.Sc.
Chair of Production and Operations Management

Dynamic Programming Basics

Components
(N, $\mathbb{X}$, $\mathbb{A}$, p, r, V_N, α)

Decision rule $f_n(x)$
Policy π
Value function $V_n(x)$

Optimality Equations

Under some weak assumption, the value function is the unique solution to the optimality equation

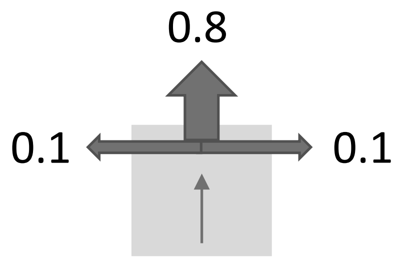
$$V_n(x) = \max_{a \in \mathbb{A}_n(x)} \left\{ r_n(x,a) + \alpha \sum_{x' \in \mathbb{X}_{n+1}} p_n(x,a,x')\, V_{n+1}(x') \right\}$$

for all $x \in \mathbb{X}_n$ and n=0,…,N-1, which can be obtained for n=N-1,…,0 iteratively, starting with $V_N(x)$. Every policy π^* consisting of actions a = $f_n^*(x)$ maximizing the right hand side of the equation is optimal.

(This weak assumption is fulfilled if the state space $\mathbb{X}$ is finite or the one-stage reward and the terminal reward are bounded by a constant.)

Grid world – infinite-horizon

$r = -0.04$, $\alpha=1$
$V_T(x)=0$
stochastic movements



0.812	0.868	0.918	+1
0.762		0.660	-1
0.705	0.655	0.611	0.388

value function results

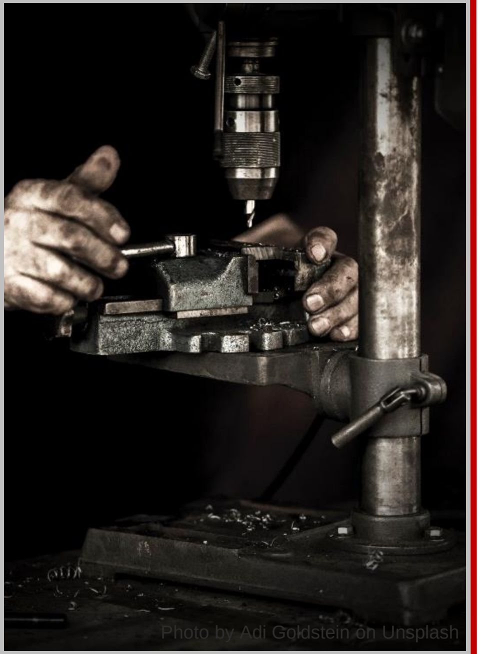
→	→	→	+1
↑		↑	-1
↑	←	←	←

optimal policy

→ choose the action that maximizes the expected value of the subsequent state:

$$\pi^*(x) = \arg\max_{a \in \mathbb{A}(x)} \sum_{x'} p(x,a,x')\, V(x')$$

In Example 1: Machine Replacement I



Optimality Equations

$$V_n(x) = \max_{a \in \mathbb{A}_n(x)} \left\{ r_n(x, a) \right.$$

In Example 1: Machine Replacement II

Components

(N, $\mathbb{X}$, $\mathbb{A}$, p, r, V_N, α)

Decision rule $f_n(x)$

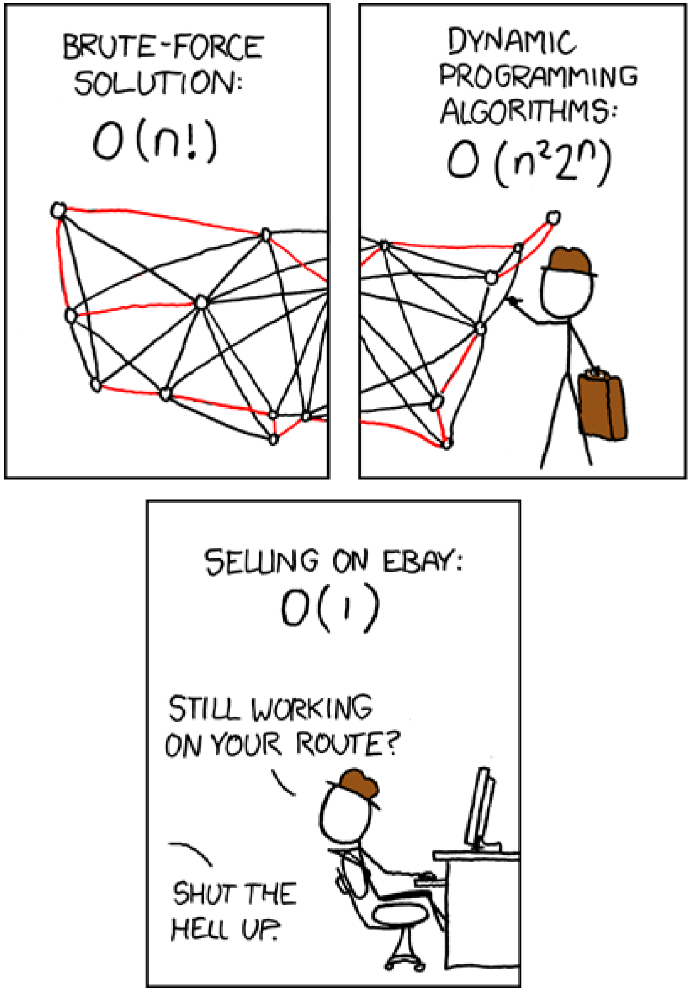
Policy π

Value function $V_n(x)$

Optimality Equations

$$V_n(x) = \max_{a \in \mathbb{A}_n(x)} \left\{ r_n(x, a) \right.$$

2. Basics of Dynamic Programming



Picture https://xkcd.com/399/ (The travelling salesman problem)

a. Introduction
   - Games to play
   - A Machine Replacement Model
   - Predictive Maintenance

b. Dynamic Programming Basics
   - Components of a Stochastic Dynamic Program (Markov Decision Process)
   - The Optimality Equations

c. Solving the Optimality Equations

Case Study 3: Sea Crest B&B

Components
(N, $\mathbb{X}$, $\mathbb{A}$, p, r, V_N, α)

Decision rule $f_n(x)$
Policy π
Value function $V_n(x)$

Optimality Equations

$$V_n(x) = \max_{a \in \mathbb{A}_n(x)} \Big\{ r_n(x,a)$$

Solving the Optimality Equations

1. **Backward Induction (Value Iteration)**
   - Determine $V_N(x)$ for all $x \in \mathbb{X}_N$
   - For all n = N-1,…,0:
     - For $x \in \mathbb{X}_n$
       - Evaluate the value function $V_n(x)$ using the optimality equations;
       - Let a* be the action that maximized the right hand side of the optimality equation.
       - Then, $f_n(x) = a^*$.

Solving the Optimality Equations

Components
(N, $\mathbb{X}$, $\mathbb{A}$, p, r, V_N, α)

Decision rule $f_n(x)$
Policy π
Value function $V_n(x)$

Optimality Equations

$$V_n(x) = \max_{a \in \mathbb{A}_n(x)} \left\{ r_n(x,a) \right.$$

2. Linear Programming

Solve:

$$\min \sum_{n=0}^{N} \sum_{x \in \mathbb{X}_n} w_{n,x}$$

s.t.

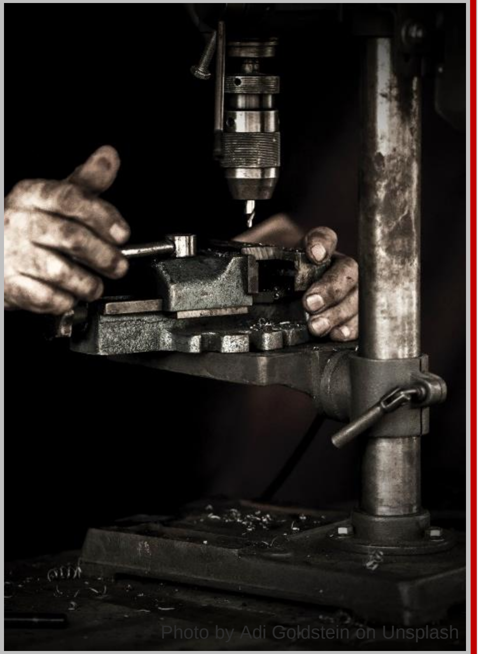
$w_{n,x} \geq r_n(x,a) + \alpha \sum_{x' \in \mathbb{X}_{n+1}} p_n(x,a,x') w_{n+1,x'}$ for all n<N, $x \in \mathbb{X}_n$, $a \in \mathbb{A}_n(x)$

$w_{N,x} \geq V_N(x)$ for all $x \in \mathbb{X}_n$

- Let $V_n(x) = w_{n,x}$
- Let $a^* \in \mathbb{A}_n(x)$ be the action that corresponds to a condition with a slack of 0. Then, $f_n(x) = a^*$.

In Example 1: Machine Replacement

Solving Dynamic Programs



Optimality Equations

$$V_n(x) = \max_{a \in \mathbb{A}_n(x)} \left\{ r_n(x, a) \right.$$

In Example 1: Machine Replacement

GAMS Code

ord() returns position of a member in a set; the first element will be 1

card() returns the number of members in a set

```
sets
n time periods /0*3/,
x age range /0*10/,
xn(x) age start n /1*5/ ;

parameter
b costs of new machine /65/ ;

parameter
c(x) maintenance costs /
0 10
1 20
2 33
3 50
4 70
5*10 1000/ ;

parameter
s(xn) salvage value /
1 30
2 15
3 10
4 5
5 0/ ;

variable
z    objective value,
w(xn,n) expected value starting in state x at n;

equations
objective
condition_replace
condition_keep
condition_terminal;
objective..                                 z =e= sum( (xn,n), w(xn,n));
condition_replace(xn,n)$(ord(n) < card(n))..  w(xn,n) =g= - c('0') - b + w('1',n+1);
condition_keep(xn,n)$(ord(n) < card(n))..     w(xn,n) =g=  - c(xn) + w(xn+1,n+1);
condition_terminal(xn,n)$(ord(n) = card(n))..  w(xn,n) =g= s(xn);

model replacement /all/;
solve replacement minimizing z using lp;

display z.l, w.l;
```

In Example 1: Machine Replacement

GAMS Code
Analyze Output

```
----      44 VARIABLE z.L                   =    -1029.000  objective value

----      44 VARIABLE w.L  expected value starting in state x at n

             0           1           2           3

1      -98.000     -43.000      -5.000      30.000
2     -113.000     -78.000     -23.000      15.000
3     -118.000     -80.000     -45.000      10.000
4     -118.000     -80.000     -45.000       5.000
5     -118.000     -80.000     -45.000
```

```
---- EQU condition_replace

           LOWER       LEVEL       UPPER     MARGINAL

1.0      -75.000     -55.000      +INF          .
1.1      -75.000     -38.000      +INF          .
1.2      -75.000     -35.000      +INF          .
2.0      -75.000     -70.000      +INF          .
2.1      -75.000     -73.000      +INF          .
2.2      -75.000     -53.000      +INF          .
3.0      -75.000     -75.000      +INF        1.000
3.1      -75.000     -75.000      +INF        2.000
3.2      -75.000     -75.000      +INF        3.000
4.0      -75.000     -75.000      +INF        1.000
4.1      -75.000     -75.000      +INF        1.000
4.2      -75.000     -75.000      +INF        1.000
5.0      -75.000     -75.000      +INF        1.000
5.1      -75.000     -75.000      +INF        1.000
5.2      -75.000     -75.000      +INF        1.000
```

```
---- EQU condition_keep

           LOWER       LEVEL       UPPER     MARGINAL

1.0      -20.000     -20.000      +INF        1.000
1.1      -20.000     -20.000      +INF        4.000
1.2      -20.000     -20.000      +INF        5.000
2.0      -33.000     -33.000      +INF        1.000
2.1      -33.000     -33.000      +INF        2.000
2.2      -33.000     -33.000      +INF        5.000
3.0      -50.000     -38.000      +INF          .
3.1      -50.000     -35.000      +INF          .
3.2      -50.000     -50.000      +INF          .
4.0      -70.000     -38.000      +INF          .
4.1      -70.000     -35.000      +INF          .
4.2      -70.000     -45.000      +INF          .
5.0    -1000.000    -118.000      +INF          .
5.1    -1000.000     -80.000      +INF          .
5.2    -1000.000     -45.000      +INF          .
```

```
---- EQU condition_terminal

           LOWER       LEVEL       UPPER     MARGINAL

1.3       30.000      30.000      +INF        6.000
2.3       15.000      15.000      +INF        6.000
3.3       10.000      10.000      +INF        6.000
4.3        5.000       5.000      +INF        1.000
5.3          .           .        +INF        1.000
```

Group Work

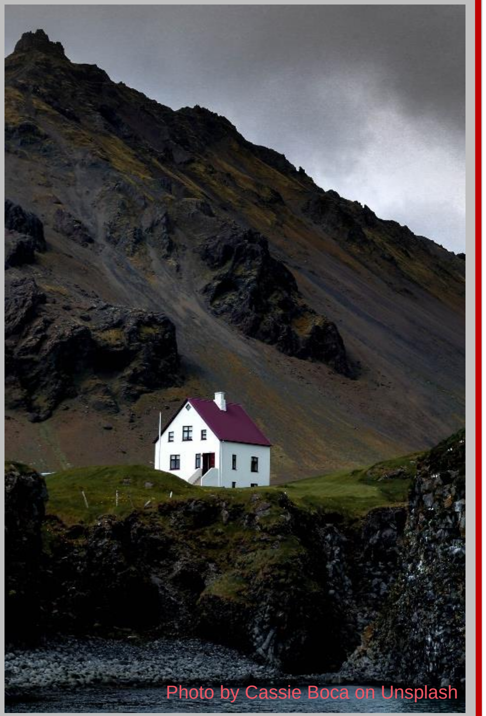


Photo by Cassie Boca on Unsplash

Case 3: Sea Crest B&B

Read the case study and answer the following question:

1. Formulate the decision problem as a finite-horizon Markov Decision Process. Determine the components: N, $\mathbb{X}$, $\mathbb{A}$, p, r, V_N, α .

(i) N

(ii) $\mathbb{X}_n$

(iii) $\mathbb{A}_n$

(iv) p_n

(v) r_n

(vi) V_N

(vii) α

Group Work

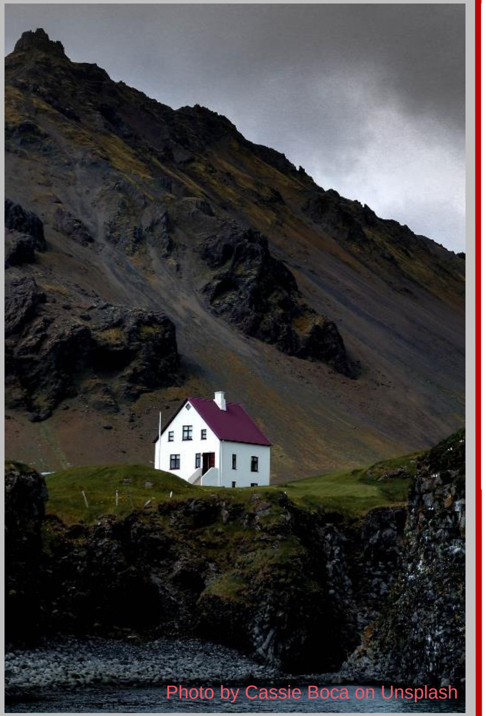


Photo by Cassie Boca on Unsplash

Case 3: Sea Crest B&B

Read the case study and answer the following question:

2. Formulate the optimality equations.

Group Task : GAMS + Excel

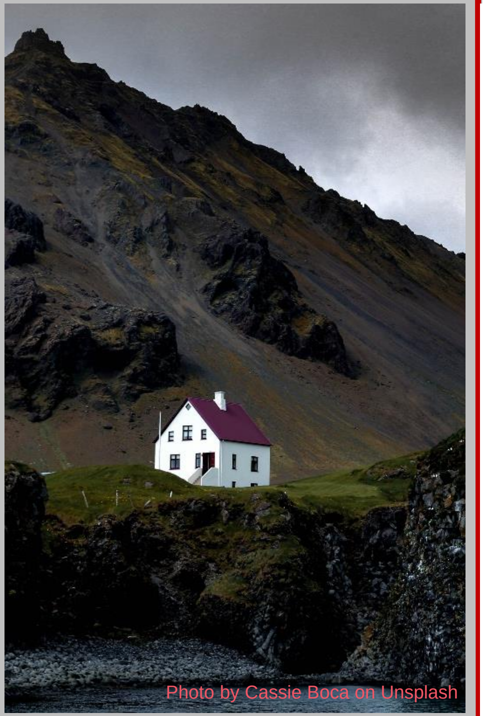


Photo by Cassie Boca on Unsplash

Use:

171023_stopping.gms
171023_Case3_SeaCrest_Template.xls

Case 3: Sea Crest B&B

3. Determine the optimal policy using GAMS.

   Hint: Adapt the Stopping Problem Code.

4. Suppose that the family agrees to accept the first offer of $3 million dollars or more (or 1 million if the first 10 offers were unacceptable). Develop a simulation to estimate the expected value and the standard deviation of the net proceeds (selling price less the cost of the consultant). Use a sample size of 500.

   Use: Excel Table “171023_Case3_SeaCrest_Template” , sheet “SimulateSeacrest”

4. Calculate a 95% confidence interval for the expected net proceeds you obtained.

5. Some members of the Crest family argue that $3 million is too low, since even the expected value of one month is higher than that. This part of the family believes that no offer less than $4.5 million should be accepted. Use your simulation to help them estimate the expected net proceeds and standard deviation of net proceeds for this proposed decision rule.

6. What would you advise the Crest family to do? Compare your GAMS solution to the solution path in “BestDecisions”. Compare the simulation results to task 4 and 6.

Typical Applications of finite-horizon MDPs

- Selling perishable products or services
- Production planning
- Patient admission and scheduling (hospitals)
- Assigning workers to incoming orders
- Queueing models (often modelled over an infinite horizon)
- Search problems

Dynamic Programming

- Formulation as a finite-horizon Markov Decision Process containing all the necessary components
- Formulation of the optimality equations (Bellman equation)
- Solving the problem by hand via backward induction (value iteration)
- Formulation of the according Linear Problem
- Understanding GAMS Code and analyzing output to find the optimal decision rule
- Using simultation to evaluate different decision rules