Get Me Out of Here: Determining Optimal Policies

Introduction

- The game Labyrinth
- Forms a maze
- Fixed and moving pieces
- Players rearrange maze to advantage by moving row
- Player has to find treasures and return to goal tile
- Motivation for my essay: How to exit the labyrinth as fast as possible
- Therefore dynamic programming:
- Given: grid map M = Znm2 containing static walls M(i ; j) = 1 and
- exit(s) of the labyrinth, set of actions A = f ;!; "; #g
- Determine optimal policy P = Znm
- 2 ! A to exit the labyrinth

Fundamentals

Was ist Dynamic Programming?

- Mathematical optimization and computer programming method that transforms complex problem into sequence of simpler problems
- Multistage nature of optimization procedure, means:
    - Solve a problem by breaking down into subproblems & recursively finding general optimal solution by solving local optimal solution of subproblems
- But provides a general framework for analyzing many problem types
    - Planning of industrial production lines
    - Scheduling of patients at a medical clinic
    - Long term investment programms

Short example

- Imagine beeing in a house and there are 2 walls
- Behind 1 wall there is a cake, your goal (blue)
- Looking like:
- When solving with dynamic programming, you can determine the best action for every position, looking like:
- Insert table with example
- Therefore: the optimal policy is beeing backward iterated starting at the goal and backward calculating always the lowest possible value for every tile
    i. Richard Bellman on the Birth of Dynamic Programming
        1. https://pubsonline.informs.org/doi/pdf/10.1287/opre.50.1.48.17791
    ii. Short example
        1. https://www.quora.com/How-should-I-explain-dynamic-programming-to-a-4-year-old/answer/Jonathan-Paulson
        2. Fibonnaci

A bit about the inventor, Richard Bellman

- But why is this working and always returning an optimal policy?

- Therefore looking at the birth of dynamic programming
- Made by Richard Bellman, an US-american mathematican
- From 1949 till 1955 out of his interest in multistage decision problems
- Recieved his Ph.D. from Princeton at the age of 25
- When he starting his research at RAND Corporation (Research And Development) he had a Secretary that really hated the word research
- So he searched for a name under which he could hide his actual mathematical research
- He was interested in planning, decision making, thinking, but none of these were good terms
- So he decided on the word programming
- And because of his love for physics and his ideas in multistage and time varying stuff, he came to the conclusion that dynamic is the best adjective to accompany the word programming
- So the name dynamic programming was born, and noone could imaging what he was doing and it was perfect for hiding his mathematicl research

Where does it come from? The Principle of optimality

- The theory of dynamic programming P.8
- For terminology
  - Policy = Sequence of decisions
  - Optimal policy = Policy that is best wrt. Some criterion
- Classically: compute all possible policies and maximize return to determine optimal
- But, often not practical
  - Even low number of stages and choices the dimension of maximation problem will explode
- Bellman saw:
  - No requirement of knowledge of complete sequence of decisions, present, next…
  - Better find a general prescription which gives a decision at any time
  - Depending only on current state of system
  - And resulting:
    - "If at any particular time we know what to do, it is never necessary to know the decisions required at subsequent times."[1]
- That really reduces the dimension of the problem to a proper level

The fundamental approach

- From the previous statements, Bellman went over to viewing an optimal policy
- Is determining the decision at each time in terms of the current state of the system
- Thereof resulting, his Principle of Optimality was made:
  - "An optimal policy has the property, that whatever the initial state and iinitial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions."[2]

Mathematical Formulation

- But what does this mean for the functional equations needed for actual problem solving based on the principle of optimality
- Imagine the simplest case, where a process of a system is described at any time by an M-dimensional vector p = (p_1, p_2,…,p_M) \elem D, D is some region
- Let T = {T_k} be a set of transformations with property p \elem D => T_k(p) \elem D \forall k

[1] The Theory of Dynamic Programming, Richard Bellman, 1954, P.3, L.12.ff
[2] The Theory of Dynamic Programming, Richard Bellman, 1954, P.4, L.9.ff

- Now thinking of an N-stage process, we want to maximize a scalar function, R(p) of the final state ^= N-stage return
- A policy is a selection of N transformations, P = (T_1, T_2, … , T_N)
- Thereof states arise:
    - p_1 = T_1(p)
    - p_2 = T_2(p_1)
    - …
    - p_N = T_N(p_N-1)
- If D is finite and if each T_k(p) is continuous in p, and if R(p) is a continuous function of p for p \elem D => An optimal policy exists
- An optimal policy will now determine the maximum value of R(p_N)
- Descibed by a function only of the initial vector p and the number of stages N
- This resulted in a function for the N-stage return coming from an optimal policy starting from initial state p
    - F_N(p) = Max_P R(p_N)
- After applying the principle, we can now derive a functional equation for f_N(p)
- When choosing a transformation T_k, resulting from first decision, leads us to a new state T_k(p)
- The maximum return from the next (N-1) stages is, as defined above f_N-1(T_k(p))
- The maximize the overall function, it follows that k must be chosen to maximize this
- Resulting
    - F_N(p) = Max_k f_N-1(T_k(p)), N=2,3,…
- From that, for any particular optimal policy, will yield f_N(p)
- On the other side, given the sequence {f_N(p)}, all optimal policies may be determined

The use of Dynamic Programming in computer science

What does it need for a problem in computer science to be solvable by a dynamic programming algorithm?

Overlapping subproblems

- Means, that space of sub-problems is small, and the problem can be broken down into subproblems
- Reuse them several times or an algorithm solves the same sub-problem over and over again.
- Again, remember the labyrinth.
- If there is only one exit and it lays at the end of a long and narrow passage.
- If you calculate the shortest paths, you will always end up walking down that narrow passage
- So, if you calculate the paths you will end up calculating the way down this passage over and over again

Optimal Substructure

- Solution to a given problem from
- Combination of optimal solutions to its subproblems
- Normally greedy algorithms used to solve problems with optimal substructure
- Find global solution for problem by finding optimal solution for subproblem
- The principle of optimality is based on this idea
- Start at some point in time t and reach end T
- Solve subproblem at all stages s where t<s<T to determine optimal policy
- Remember the labyrinth for example.
- For any given position p in the labyrinth a shortest path s around the walls to the nearest exit e can be calculated.

- If s is really the shortest path, can be split into subpaths s_1 from p to x and s_2 from x to e such that these are indeed the shortest paths between the three positions
- From that, a recursive way for finding shortest paths can easily be formulated, which is what the Bellman-Ford algorithm does

How to really use it

Memoization

- Dynamic programming due to optimal substructure
- Breaking down into set of simpler subproblems
- Solve each subproblem only once and save result in table..
- When a subproblem occurs again, return cached result
- Saves computation time, but expansion in storage
- Mostly neglegtable due to rapidly expanding memory sizes and shrinking costs

Bellman equation

- Dynamic Programming equation
- Necessary condition for optimality
- Makes a dependency of the value of a decision problem at a certain point in time of the outcome of the initial choics and the value of the remaining decisions that result from the predeceasing choices.
- This breaks the problem down into subproblems that can be calculated separately, as Bellman's principle of optimality suggests
- Objective function
- Before coming to the bellman equation, some underlying concepts have to be defined.
- 1. Optimizing a problem is always under a certain objective: maximizing utility, minimizing pathlength, maximizing profits. The function describing these objectives is called the **objective function.**
- State
- Dynamic Programming needs to keep track of the **state** of the system when evolving over time, because when breaking down a multi-period planning problem into simpler steps at different points in time, in every step the information of the current situation that is needed has to be available.
- Control variables
- The next state is always affected by some factors in addition to the current **control.** That means, that choosing the control variables may be equivalent to choosing the next state, which means making a decision
- In dynamic programming, an optimal plan can now be described by finding a rule that makes an interconnection between what the controls should be when given any possible value of the state.
- For now achieving the best possible value of the objective, the optimal decision rule has to be found.
- For example
- ~~In the maze, if you choose direction, given the current path length, in order to minimize the shortest path length, then each level of current length can be associated with~~
-
- Finally, by definition, the optimal decision rule is the one that achieves the best possible value of the objective. When written as a function of the state, the best possible value of the objective is called the **value function**
- Richard Bellman showed, that when calculating a relation between the value function of one step and the value function of the next step, a dynamic optimization problem can be solved

in a recursive manner. The **Bellman equation** is now just the relationship connecting these two value functions.

- Coming directly from this equation, Bellman gave a method for applying it onto some concrete problem, known as backward induction. In this approach, the optimal policy of the last time period (t = N) is specified at first as a function depending on the value of this state and the optimal value of the objective function.
- Next, the previous step in time (t = N-1) is calculated by optimizing the current objective function and the optimal value for the future objective function.
- This can be applied recursively until the initial state (t = 0) is reached until the first period decision rule is derived, as a function of the initial state and the value, by optimizing the sum of the objective function of the state at the initial time step (t = 0) and the value of the second timestep (t = 1), which is dependent on all future decisions.
- *Resulting from that, each periods decision is made by explicitly acknowledging that all utur decisions will be optimally made.*
- So, when used on a dynamic decision problem the function describing the previously stated thoughts is
- V(x_0) = max_a_0 {F(x_0, a_0) + V(x_1)}
- Whereas x_t is the state of the system at time t, beginning at 0, initial state x_0
- Whereas a_0 \in tau(x_0) is the action depending on the set of possible actions tau
- The transition from one state x to another, is described by T(x,a), so x_1 = T(x_0, a_0)…
- when action a is chosen at state x, and the payoff of taking a in state x is F(x,a),
- So when dropping the time dependency, the bellman equation can be simplified to
- V(x) = max_a\in\tau(x) {F(x,a) + V(T(x,a))}
- Solving this functional equation means now finding the unknown function V, which is the value function

- Consider the maze with nxm squares
- And a cost function c(i,j) which returns the cost associated with the square in column x, row y
- Now consider a set of tuples (I,j) that represent goal states
- Now imagine being in the maze at any arbitrary starting position p and you wanted to know the optimal policy to determine the shortest path to the nearest goal
- For moving, you can only go right, down, left, up
    - So being at (2,2) you could go to (2,3),(3,2),(2,1),(1,2)
- This problem exhibits optimal substructure
- Entire problem relies on solution to subproblems
- Let define a value function v(I,j) as v(i,j) = the minimum cost to reach square (i,j)
- If one can find the values of this function for all squares, one can pick the minimum and follow path to get shortest path
- Thereof, one can derive a general and recursive function for v(I,j)
- V(I,j) = {
    - \inf   if i<1, i>n or j<1, j>n
    - 0   if (I,j) \in goals
    - Min(v(i, j+1) + c(I, j+1), v(i+1, j) + x(i+1,j), v(I, j-1) + c(I,j-1), v(i-1, j) + c(i-1,j))   else,
    - }
- First line is to check if out of Maze
- Second is base case for stop recursion
- Last is recursion step
- This will only calculate cost of shortest path, later I will provide pseudo code that is slightly different implemented for calculating actual optimal policy

How to apply it, Write out if space left

Top down approach

Bottom up approach

Related Work

Classic Types of Motion Planning, Short Introduction

Sampling based Motion Planning

- Maybe later

Combinatorial Motion Planning

- Maybe later

Pathfinding in 2D Grid

Breadth First

- Maybe later

A*

- Maybe later

Dynamic Programming

Bellman Ford Algorithm for optimal policies

-

3. https://www-m9.ma.tum.de/graph-algorithms/spp-bellman-ford/index_de.html

iii. Usage of Dynamic Programming in Motion Planning

1. Feeback Motion Planning, LaValle, Planning Algortihms
    a. http://planning.cs.uiuc.edu/
2. RRT#, RRT* Algorithm, Arslan Dissertation
    a. Machine learning and dynamic programming algorithms for motion planning and control
    b. https://smartech.gatech.edu/handle/1853/54317
3. Technical Note—Determining All Optimal and NearOptimal Solutions when Solving Shortest Path Problems by Dynamic Programming
    a. Byers, Waterman
    b. https://pubsonline.informs.org/doi/pdf/10.1287/opre.32.6.1381
4. From Dynamic Programming to RRTs: Algorithmic Design of Feasible Trajectories
    a. http://msl.cs.illinois.edu/~lavalle/papers/Lav02.pdf
5. ON A ROUTING PROBLEM
    a. Richard Bellman
    b. http://www.ams.org/journals/qam/1958-16-01/S0033-569X-1958-0102435-2/
6. The Shortest Route Through a Network with Time-Dependent Internodal Transit Times
    a. KENNETH L. COOKE AND ERIC HALSEY
    b. http://www.sciencedirect.com/science/article/pii/0022247X66900096

            7. benjamin gutjahr, IEEE
        iv. Comparison of Dynamic Programming
            1. Differences to other Algorithms
            2. Why is DP good for Motion Planning

2. Solution Approach
    a. Motivation
    b. Topic
    c. My task
    d. My implementation
3. Evaluation
    a. Correctness with A*
    b. Laufzeit
    c. Hoffentlich mathematischer Beweis
4. Conclusion
    a. Dynamic programming different approach classic algorithms
    b. Great way for time improvements
    c. But space tradeoff
    d. Hard to master
    e. But when mastered really powerful way to think of problems and tasks