Dynamic Programming

Richard de Neufville
Professor of Engineering Systems
and of
Civil and Environmental Engineering
MIT

Objective

- To develop formally "dynamic programming", the method used in lattice valuation of options
  - Assumptions: Separability and Monotonicity
  - Its optimization procedure: "Implicit Enumeration"
- To show its wide applicability
  - Options analysis
  - Sequential problems: routing and logistics; inventory plans; replacement policies; reliability
  - Non-sequential problems: investments

Outline

1. Why in this course?
2. Basic concept: Implicit Enumeration
   – Motivational Example
3. Key Assumptions
   – Independence (Separability) and Monotonicity
4. Mathematics
   – Recurrence Formulas
5. Example
6. Types of Problems DP can solve
7. Summary

Why in this course?

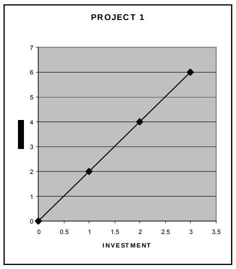
- DP is used to find optimum exercise of options in lattice, that has non-convex feasible region (see comments in that presentation)

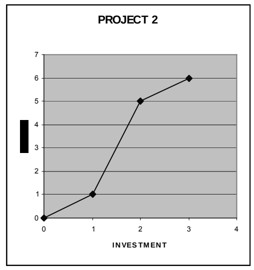
- This presentation gives general method, so you understand it at deeper level

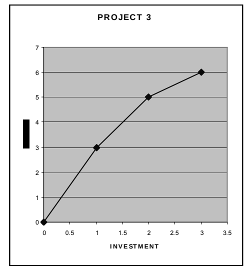
- DP used in lattice is simple version
  – only 2 states considered compared at any time

Motivational Example

Consider Possible Investments in 3 Projects



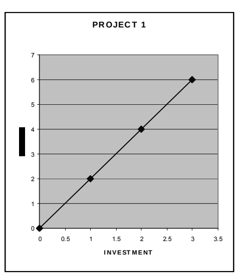


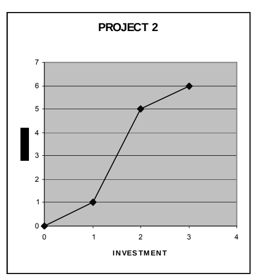


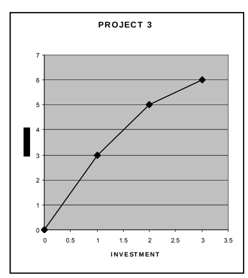
What is best investment of 1st unit? P3 ➔ +3

Of 2nd? 3rd? P1 or P3 ➔ +2, +2 Total = 7

Motivational Example: Best Solution







Optimum Allocation is Actually (0, 2, 1) ➔ 8

Marginal Analysis misses this….
because Feasible Region is not convex

Point of Example

- Non-convexity of feasible region can "hide" optimum solution
- Marginal analysis ,"hill climbing" methods to search for optimum not appropriate in these cases (for lattice models in particular)

- We need to search entire space of possibilities
- This is what "Dynamic Programming" does to define optimum solution

Semantic Note

- "Dynamic" Programming so named because
  - Originally associated with movements through time and space (e.g., aircraft gaining altitude, thus "dynamic")
  - "programming" by analogy to "linear programming" and other forms of optimization

- This approach can be used in many cases that are not in fact "dynamic"

Basic Solution Strategy

- Enumeration is basic concept
  - This means evaluating "all" the possibilities
  - Checking "all" possibilities, we must find best
- No assumptions about regularity of Objective Function
- Means that DP can optimize over
  - Non-Convex Feasible Regions
  - Discontinuous, Integer Functions
  - Which other optimization techniques cannot do
- HOWEVER…

Curse of Dimensionality

- Number of Possible Designs very large
- Example: a simple development of 2 sites, for 4 sizes of operations over 3 periods
  - Number of Combinations in 1 period = 4^2 = 16
  - Possibilities over 3 periods = 16^3 = 4096
- In general, size of design is exponential
  $= [\ (\text{Size})^{\text{locations}}]^{\ \text{periods}}$
  - Actual enumeration is impractical
- In lattice model…. See next page

The Curse -- in lattice model

OUTCOME LATTICE

100.00	120.00	144.00	172.80	207.36	248.83	298.60
	80.00	96.00	115.20	138.24	165.89	199.07
		64.00	76.80	92.16	110.59	132.71
			51.20	61.44	73.73	88.47
				40.96	49.15	58.98
					32.77	39.32
						26.21

- **End states = N**
- **Total States ~ Order of only N^2 /2**
- **Number of paths ~ Order of 2^N…**
  - **To reach each state at last stage = 1 + 6 +13 +16 + 13 + 6 +1 = 46 paths**

Implicit Enumeration

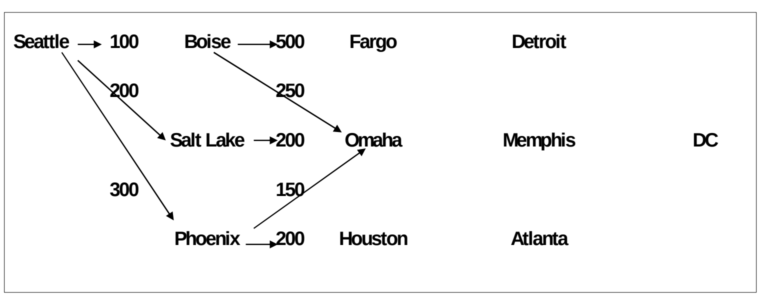
- **IE considers all possibilities in principle**
- **Exploits features of problem to**
  - **Identify classes of dominated possibilities**
  - **Reject these classes**
  - **Vastly reduce dimensionality of enumeration**
- **Size of numeration for DP**
  - **Order of [(Size) Locations] Periods**
  - **Multiplicative size, not exponential**
  - **This analysis computationally practical**
- **Examples will illustrate what this means**

Demonstration of I E

- Select a “dynamic” problem – logistic movement from Seattle to Washington DC
- Suppose that
  - there are 4 days to take trip…
  - Can go through several cities
  - There is a cost for the movement between any city and possible city in next stage
- What is the minimum cost route?

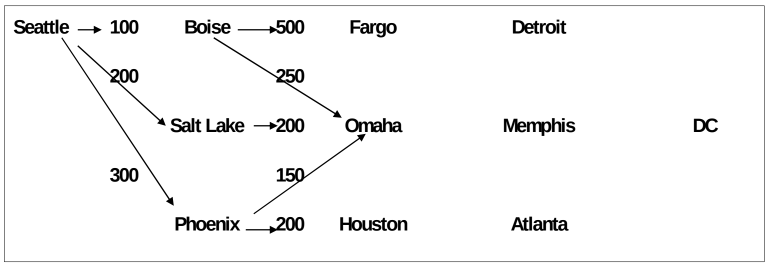
Possible routes through a node

- Many routes, with link costs as in diagram
- Consider Omaha
  - 3 routes to get there, as shown
  - 3 routes from there => 9 routes via Omaha



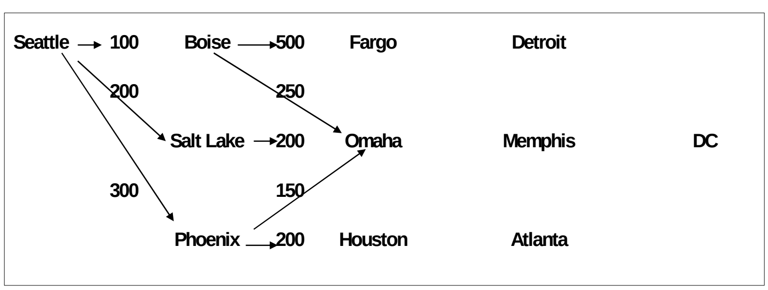
Instead of Costing all Routes… IE

- We find best cost to Omaha (350 via Boise)
- Salt Lake (400), Phoenix (450) routes dominated, "pruned" – we drop routes with those segments
- Thus don't look at all Seattle to DC routes



Result: Fewer Combinations

- Total Routes:
  = 3 to Omaha + 3 after = 6 = 3 x 2, not 9 = 3^2
- Savings not dramatic here, illustrate idea



Some nomenclature definitions

- In Dynamic Programming:
- The Objective Function is $G(\underline{X})$
- Where $\underline{X} = (X_1, \ldots. X_N)$ is the set of states at each of N "stages" --
- Selection of all X_i defines optimum policy
- $g_i X_i$ are the "return functions" that provide the effect of an X_i state at the i^{th} stage
- $g_i X_i$ denotes the functional form; X_i the different states at the i^{th} stage
- So that $G(\underline{X}) = [g_1 X_1, \ldots. g_N X_N]$

Useful Concepts -- Stages

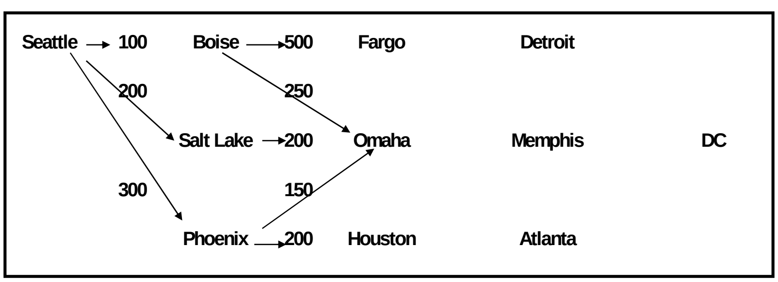
- Each $g_i X_i$ "return function" is associated with a "stage" in the problem
  - Example: 1^{st} Stage is from Seattle to Boise, etc
  - Thus $g_1 X_1$ are costs from Seattle to Boise, etc
- "Stages" may
  - have a physical meaning, as in example,
  - or be conceptual (as the investments in later example) where a "stage" represents the next project or "knob" for system that we address

Useful Concepts -- States

- **Each g_iX_i takes on different "states", that is, possible situations at a stage**
- **Examples:**
  - **For cross-country shipment, there are 3 states (of system, not as "states" of USA) for 1st stage, Boise, Salt Lake and Phoenix**
  - **For plane accelerating to altitude, a state might be defined by (speed, altitude) vector**
  - **For investments, states might be $ invested**
  - **If stage is "knob" we manipulate on system, "state" is the setting of the knob**

Stages and States

- **"Stages" are associated with each move along trip**
- **Stage 1 consists of set of endpoints Boise, Salt Lake and Phoenix, Stage 2 the set of Fargo, Omaha and Houston; etc.**
- **"States" are possibilities in each Stage: Boise, Salt Lake, etc...**



Solution depends on Decomposition

- **Must be able to "decompose" objective function G($\underline{X}$) into functions of individual "stages" X_i:**

  $G(\underline{X}) = [g_1X_1, \ldots.\ g_NX_N]$

  - Example: cost of Seattle to DC trip can be decomposed into cost of 4 segments of which Seattle to Boise, Salt Lake or Phoenix is first
- **Necessary conditions for decomposition**
  - Separability
  - Monotonicity

Separability

- **Objective Function is separable if all g_iX_i are independent of g_JX_J for all J not equal to I**
- **In example, it is reasonable to assume that the cost of driving between any pair of cities is not affected by that between another pair**
- **However, not always so…**

Monotonicity

- Objective Function is monotonic if: improvements in each $g_i X_i$ lead to improvements in Objective Function, that is if
- given $G(\underline{X}) = [g_i X_i , G'(\underline{X'})]$ where $\underline{X} = [X_i , \underline{X}']$
- for all $g_i X'_i \geq g_i X''_i$ where X'_i , X''_i different X_i
- It is true that $[g_i X'_i , G'(\underline{X})] \geq [g_i X''_i , G'(\underline{X})]$
- Additive functions always monotonic
- Multiplicative functions monotonic only if $g_i X_i$ are non-negative, real

Solution Strategy

- Two Steps
  - Partial optimization at each stage
  - Repetition of process for all stages
- This is the process used to value options through the lattice
  - At each stage (period), for each state (possible outcome for system)
  - The process chooses better of exercising option or not

Cumulative Return Function

- Result of Optimization at each stage and state is the "cumulative return function"
- f_S (K) denotes best value for being in state K, having passed through previous S stages
- Example: f_2 (Omaha) = 350
- Defined in terms of best over previous stages and return functions for this stage:

$$f_S(K) = \text{Max or Min of } [g_i X_i \,,\, f_{S-1}(K)]$$

(note: K understood to be a variable)

Mathematics: Recurrence formulas

- Transition from one stage to next is via a "recurrence formula" (or equivalent analysis)
- Formally, we seek the best we can obtain to any specified level K*, by finding the best combination of possible $g_i X_i$ and f_{S-1} (K)
- This is what we did for the valuation of options in the lattice:

  = NPV[r, Max[EV(mine open), cost of closing]]
- Where value of "mine open" is immediate value ($g_i X_i$) and later stages, f_{S-1} (K)

Application of Recurrence formulas

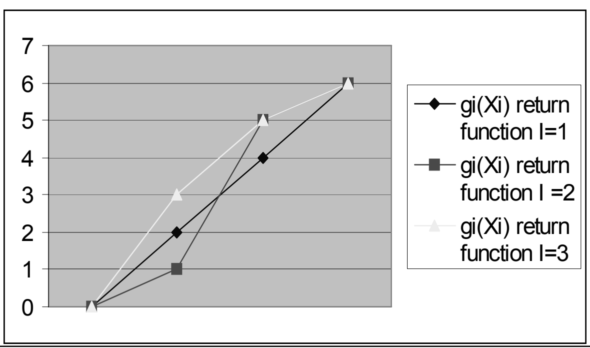
- **For Example: Consider the Maximization investments in independent projects**
  - Each project is a "stage"
  - Amount of Investment in each is its "state"
  - Objective Function Is Additive:

    Value = Σ (value each project)
  - Recurrence formula: $f_i (K) = Max[g_i X_i + f_{i-1} (K- X_i)]$
  - that is: optimum for investing K over "i" stages

  = maximum of all combinations of investing level X_i in stage "i" and (K- X_i) in previous stages

Application to Investment Example

- **3 Projects, 4 Investment levels (0, 1, 2, 3)**
- **Objective: Maximum for investing 3 units**
- **Stages = projects ; States = investment levels**



Dynamic Programming Analysis (1)

- At 1st stage the cumulative return function identically equals return for X_1
- That is, $f_1(X_1)$, the best way to allocate resource over only one stage $\equiv g_1X_1$
  - There is no other choice
- So $f_1(0) = 0$
- $f_1(1) = 2$; $f_1(2) = 4$; $f_1(3) = 6$

Dynamic Programming Analysis (2)

- At 2nd stage, best way to spend:

  0 : is 0 on both 1st and 2nd stage (= 0) = $f_2(0)$

  1 : either: 0 on 1st and 1 on 2nd stage (= 1)
  or: 1 on 1st and 0 on 2nd stage (= 2) BEST = $f_2(1)$

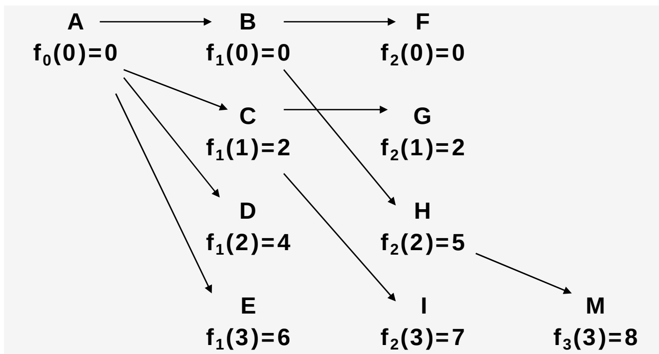
  2 : 2 on 1st , and 0 on 2nd stage (= 4)
  1 on 1st, and 1 on 2nd stage (= 3)
  0 on 1st, and 2 on 2nd stage (= 5) BEST = $f_2(2)$

  3: 4 Choices, Best allocation is (1,2) ➔ 7 = $f_2(3)$

  These results, and the corresponding allocations, shown on next figures…

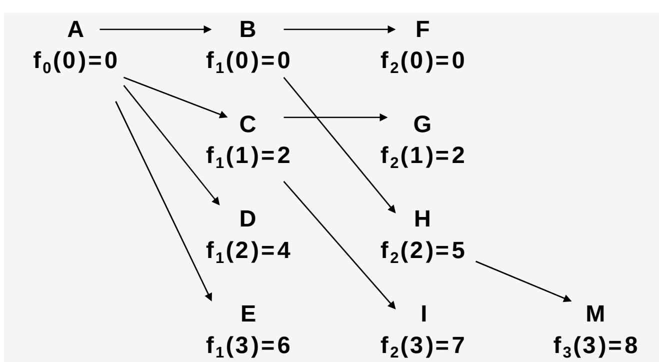
Dynamic Programming Analysis (3)

- LH Column: 0 in no Project $f_0(0) = 0$
- 2nd Column: 0…4 in 1st project, e.g.: $f_1(2) = 4$



Dynamic Programming Analysis (4)

- For 3rd stage (all 3 projects) we want optimum allocation of all 4 units: ➔ (0,2,1) ➔ $f_3(4) = 8$



Contrast DP and Marginal Analysis

- **Marginal Analysis:**
  - reduces calculation burden by only looking at "best" slopes towards goal, discards others
  - Misses opportunities to take losses for later gains
  - approach ➔ 7
- **Dynamic Programming :**
  - Looks at "all" possible positions
  - But cuts out combinations that are dominated
  - Using independence return functions (value from a state does not depend on what happened before)

Classes of Problems suitable for DP

- **<u>Sequential</u>, "Dynamic" Problems**
  -- aircraft flight paths to maximize speed, altitude
  -- movement across territory (example used)
- **Schedule, Inventory (Management over time)**
- **Reliability -- Multiplicative example, see text**
- **Options analysis!**

- **<u>Non-Sequential</u>: Investment Maximizations**
  - Nothing Dynamic. Key is separability of projects

Formulation Issues

- No standard ("canonical") form
- Careful formulations required (see text)
- DP assumes discrete states
  - thus easily handles integers, discontinuity
  - in practice does not handle continuous variables
- DP handles constraints in formulation
  - Thus certain paths not defined or allowed
- Sensitivity analysis is not automatic

Dynamic Programming Summary

- The method used to deal with lattices
- Solution by implicit enumeration
- Approach requires
  - separability, monotonicity
- Careful formulation needed
- Useful for wide range of issues

  -- in particular for options analyses!