Chapter 1: Introduction to Operations Research

The Modelling Process

Mathematical decision-making model

- Variables → decisions
- Parameters → constants
- Measure of performance
- Constraints → restrictions

Step 1: Formulation of the model and data collection

- Defining the problem
- Developing a model
- Acquiring input data

Step 2: Solution development and validation of the model

- Developing a solution
- Testing and validating the model

Step 3: Interpretation and what-if analysis

- Analyzing the results and sensitivity analysis
- Implementing the results

Chapter 2: Linear Programming

Introduction to modelling - The Graphical Solution Method

Introduction

1) Define the decision variables
2) Formulate the objective function → minimize or maximize
3) Formulate the functional constraints
4) Formulate the sign restrictions → $x_i \geq 0$ or x_i urs

Terminology

- **Feasible** solution
  - ⇨ Solution that satisfies all the problem's constraints.
- **Optimal** solution
  - ⇨ Feasible solution that results in the largest possible objective function value when maximizing (or smallest when minimizing).

Linear Programming Formulation

Three categories of constraints:

- Limited resources: "at most"
- Minimum performance: "at least"
- Conservation: "exactly"

Assumptions and Limitations of LP

- Certainty
  - ➔ All parameters of the model are known constants.
- Linearity
  - ➔ The objective function and constraints are expressed as linear functions.

- Proportionality
  - ➔ The contribution of each activity to the value of the objective function Z is proportional to the level of the activity $x_j(c_j)$.
  - ➔ The contribution of each activity to the left-hand side of each functional constraint is proportional to the level of the activity $x_j(a_{ij})$.
- Additivity
  - ➔ Every function (in LP model) is the sum of the contributions of all individual decision variables.
  - ➔ Can be violated due: complementary or competivity
- Divisibility
  - ➔ Decision variables can have non-integer or fractional values (<-> integer programming)

The Graphical Solution Method

For linear programs with two or three variables.

Solution method

1) Determine the feasible region
   - Draw the constraints one-by-one
   - Find the set of points that satisfy all constraints
2) Determine optimal solutions
   - Draw an objective function line
     - ➢ Isoprofit line (maximization)
     - ➢ Isocost line (minimization)
   - Move the objective function line (draw parallel lines) → same slope
   - Determine optimal solution
     - ➢ Intersection objective function line and the feasible region
       → the feasible solution with the largest value is the optimal solution.

- Binding (equal) and nonbinding (unequal) constraints
- Feasible region: nonexistent, a single point, a line, a polygon or an unbounded area.
- Extreme points → optimal solution

Special cases

- Alternative or multiple optimal solutions
  - ➔ In graphical method: objective function line is parallel to a boundary constraint.
- Infeasible LP
  - ➔ LP is over-constrained → no feasible solutions → no optimal solution
- Unbounded LP
  - ➔ Many feasible solutions → no optimal solution

The Simplex Method

The Principles

Optimality test

If a **CFP solution** (corner-point feasible solution) has no *adjacent* CPF solutions that are better, then it must be an **optimal solution**.

- ➔ Choice of the next adjacent CPF solution is dependent on the rate of improvement in Z (largest rate is chosen) that would be obtained by moving along the edge.

Setting Up the Simplex Method

Basic solution

- A **basic solution** is an augmented CPF solution (and includes slack variable values).
- Two basic solutions are **adjacent** if all but one of their non-basic variables are the same.
- Moving from one basic solution to another involves switching one variable from non-basic to basic and vice versa for another variable.
  - ➔ Entering variable: non-basic to basic
  - ➔ Leaving variable: basic to non-basic

The Algebra of the Simplex Method

Rewriting the Linear Functions → Jordan-Gauss elimination: elementary row operations

The Simplex Method

Step 1:

If the problem is a minimization problem, multiply the objective function by -1.

Step 2:

If the problem formulation contains any constraints with negative RHS, multiply each constraint by -1.

Step 3:

Put the problem into **standard form**:

- Add a slack variable to each $\leq$ constraint.
- Subtract a surplus variable and add an artificial variable to each $\geq$ constraint.
- Set each slack and surplus variable's coefficient in the objective function to zero.

Step 4:

Select the origin as initial basic solution:

- Select the decision variables to be the initial non-basic variables (set equal to zero).

Step 5:

Put the problem into **canonical form**:

- Add an artificial variable to each equality constraint and each $\geq$ constraint.
- Set each artificial variable's coefficient in the objective function equal to -M, where M is a very large number.
- Each slack and artificial variable becomes one of the basic variables in the initial basic solution.
- Conditions:
  1) All right-hand sides (RHS) values are positive.
  2) Each basic variable appears in 1 constraint.
  3) Each constraint contains 1 basic variable.
  4) The coefficients of the basic variables are all equal to 1.

Step 6:

Rewrite the objective function in terms of non-basic variables only such that the entering BV can be easily determined. Hence, eliminate all BV's from this row using elementary row operations.

Step 7:

Iterations:

1) Determine entering variable
   - ⇨ variable with the most negative value in the objective row
   - ⇨ corresponding column = pivot column
2) Determine leaving variable
   - ⇨ trade ratio = RHS / (exchange) coefficient
   - ⇨ select variable with smallest trade ratio
   - ⇨ corresponding row = pivot row

3) Generate New Tableau
    - ⇨ Divide the pivot row by the pivot element
    - ⇨ Replace each non-pivot row
    - ⇨ Replace the objective row

Special cases

- Tie for the entering variable → choose arbitrarily
- Tie for the leaving variable → choose arbitrarily
- **Degeneracy**: basic variable with a value of zero = degenerated variable
- **Alternative optimal solutions**: non-basic variable with an objective row value equal to zero in the final tableau → multiple optima
- **Unboundedness**: all entries in an entering column are non-positive → no leaving variable → entering basic variable could be increased indefinitely → unbounded
- **Infeasibility**: there is an artificial variable in the optimal solution → problem is infeasible

Two-Phase Method

When using **big M-method** → split the problem and solve the problem in two phases

Phase 1: Divide the big M method objective function terms by M and drop the other negligible terms.

- Minimize Z = ∑ artificial variables
- subject to Revised constraints (with artificial variables)

Phase 2: Find the optimal solution for the real problem. Use the optimal solution of phase 1 as initial basic feasible solution for applying the simplex method to the real problem (the big M method coefficients can be dropped dependent of outcome of phase 1).

- Minimize Z = ∑ original variables
- subject to Original constraints (without artificial variables)

Duality Theory

Quick estimate of the objective function value → how to find the best possible upper bound?

Construct a linear combination of the constraints and multiply an unknown variable $y_i$with each constraint. Suppose that, instead of using our resources to produce products, we sell the available resources → what prices should we charge?

➔ Decision variables y_i: per unit charge for every resource.

Sensitivity Analysis

Sensitivity analysis, also referred to as post optimality analysis

Determine how the optimal solution in an LP is affected by changes, within specified ranges, in:

- The values of the objective function coefficients.
- The values of the right-hand side coefficients.
- The values of the exchange coefficients.

How might changes in the objective function coefficients affect the optimal solution?

The **range of optimality** of an objective function coefficient provides the range of values over which the *current solution remains optimal* (keeping all other coefficients constant).

How might a change in the right-hand side for a constraint affect the feasible region and perhaps cause a change in the optimal solution?

The **range of feasibility** of a right hand side coefficient is the range of that coefficient over which the *shadow price remains unchanged*.

Shadow price vs Dual price:

The **shadow price** is the change in the objective function if the RHS is increased by 1.
The **dual price** gives the improvement in the objective function if the constraint is relaxed by one unit (RHS + 1 for ≤ constraints; RHS - 1 for ≥ constraints).
→ For ≤ constraints they are the same.
→ For ≥ constraints they have opposite signs.

Graphical Sensitivity Analysis

Graphically, the limits of a **range of optimality** are found by changing the slope of the *objective function line* within the limits of the slopes of the binding constraint line:

- The slope of an objective function line: $Z = c_1x_1 + c_2x_2 \rightarrow -\frac{c_1}{c_2}$
- The slope of a constraint: $a_1x_1 + a_2x_2 \rightarrow -\frac{a_1}{a_2}$

Graphically the **range of feasibility** is determined by finding the values of a *right hand side coefficient* such that the same two lines that determined the original optimal solution continue to determine the optimal solution for the problem.

Sensitivity Analysis with Simplex Tableaus !!!

Change in a coefficient of a decision variable

- Decision variable is a **non-basic variable**
  - ➔ Duality Theory
- Decision variable is a **basic variable**
  - ➔ Fundamental Insight
    - Objective function coefficient → range of optimality for c_k
    - Technological coefficient → range of optimality for c_k
  - ➔ When outside the range of optimality: simplex method
- Introduction of a **new variable**
  - ➔ Consider new variable as a non-basic variable
  - ➔ Duality Theory
- Introduction of a **new constraint**
  - ➔ Check feasibility of current solution:
    - If feasible: current solution remains optimal.
    - If infeasible: add constraint to final simplex table and set slack variable as initial basic variable.

Change in a right-hand side coefficient

- **Range of feasibility**
  - ➔ Fundamental Insight

Simultaneous changes in the parameters

100% rule

Calculate for each change the percentage of *the allowable change* (increase or decrease) for that coefficient. If the sum of the percent changes in the coefficients *does not exceed 100%*, the original optimal solution will *still be optimal*. (If the sum exceeds 100%, then we cannot be sure.)

Fundamental insight

For a problem with 2 constraints:

$$R0_{FINAL} = R0_{INITIAL} + y_1 R1_{INITIAL} + y_2 R2_{INITIAL}$$
$$R1_{FINAL} = s_1' R1_{INITIAL} + s_2' R2_{INITIAL}$$
$$R2_{FINAL} = s_1'' R1_{INITIAL} + s_2'' R2_{INITIAL}$$

For a problem with 3 constraints:

$$R0_{FINAL} = R0_{INITIAL} + y_1 R1_{INITIAL} + y_2 R2_{INITIAL} + y_3 R3_{INITIAL}$$
$$R1_{FINAL} = s_1' R1_{INITIAL} + s_2' R2_{INITIAL} + s_3' R3_{INITIAL}$$
$$R2_{FINAL} = s_1'' R1_{INITIAL} + s_2'' R2_{INITIAL} + s_3'' R3_{INITIAL}$$
$$R3_{FINAL} = s_1''' R1_{INITIAL} + s_2''' R2_{INITIAL} + s_3''' R3_{INITIAL}$$

Extra example B

Question 4: **Range of feasibility and sunk costs**

- Given that aluminium is a sunk cost, what is the maximum amount the company should pay for 50 extra pounds of aluminium?
- If aluminium were a relevant cost, what is the maximum amount the company should pay for 50 extra pounds of aluminium?

Answer: **Additional explanation of the theory**

- When (resource) costs are **sunk**, the cost is not considered in the objective function.
- When (resource) costs are **relevant**, the cost is considered in the objective function.
- The shadow price reveals the increase in profit (= objective function value) if the resource availability (= the right hand side of a constraint) is increased by one unit. The shadow price is thus the maximal price that you want to pay (additionally) for an extra unit of the resource.
- If the cost is **relevant**, the cost is considered in the objective function, and hence, the shadow is the additional amount that you want to pay for one unit of the resource. Thus, the price for one unit of the resource = shadow price + variable cost (considered in the objective).
- If the cost is **sunk**, the cost is not considered in the objective function, and hence, the shadow price is the amount that you want to pay for one unit of the resource. Thus the price for one unit of the resource = shadow price + 0 (since this cost is not considered in the objective).

Answer:

- Since the cost for aluminium is a sunk cost, the shadow price provides the value of extra units of aluminium. The shadow price for aluminium is the same as its dual price (for a maximisation problem). The shadow price for aluminium is €3.125 per pound and the maximum allowable increase is 60 pounds. Since 50 is in this range, then the €3.125 is valid, and the value of 50 additional pounds is €156.25.
- If the cost of aluminium were a relevant cost, the shadow price would be the amount above the normal price of aluminium the company would be willing to pay. Thus if initially aluminium cost €4 per pound, then additional units in the range of feasibility would be worth €4 + €3.125 = €7.125 per pound.

Multi-Criteria Decision Making

Goal Programming

Solve linear programs with multiple objectives and each objective viewed as a "**goal**".
The goals themselves are added to the constraint set with $\mathbf{d_i^+}$ and $\mathbf{d_i^-}$ (deviation variables) acting as the surplus and slack variables.

- Non-preemptive Goal Programming → all the goals are of roughly comparable importance
- Preemptive Goal Programming → hierarchy of priority levels for the goals

Formulate a goal programming model

1) Decide the priority of each goal
2) Decide the weight on each goal. If a priority level has more than one goal, for each goal i decide the weight, w_i, to be placed on the deviation(s), d_i^+ and/or d_i^-, from the goal.
3) Set up the initial linear program
   - ➢ Min $(w_1 d_1^+ + w_1 d_1^-)$
   - ➢ s.t. Functional and Goal Constraints

The Transportation Problem

LP formulation special cases:

- Minimum shipping guarantee
- Maximum route capacity
- Unacceptable routes

Solution methods:

- General-purpose Simplex Method
- Transportation Simplex Method

The Transportation Simplex Method

Setting Up the Transportation Simplex Method

- The Transportation Simplex Tableau
  - ➔ No artificial variables
  - ➔ Dummy column if demand is less than supply
  - ➔ Dummy row if supply is less than demand

Finding Basic Feasible Solutions for Transportation Problems

- The **Northwest Corner** Rule
  - ⇨ Step 1: Select the variable of the northwest corner of the transportation tableau (x_{11}).
- **Minimum Cost** Method
  - ⇨ Step 1: Select the variable with the smallest shipping cost.
- **Vogel's Approximation** Method
  - ⇨ Step 1: For each row and column of the table remaining under consideration, find the *difference between the two lowest unit costs* c_{ij} still remaining in that row or column (opportunity cost). In that row or column having the *largest opportunity cost* (difference), select the variable with the *smallest remaining unit cost*.

Obtaining the Optimal Solution using the Transportation Simplex Method

1) Optimality test
   - ⇨ A basic feasible solution is optimal if and only if the reduced cost ≥ 0 of every (i,j) such that x_{ij} is non-basic.
2) Select an entering basic variable
   - Calculate the reduced costs of the non-basic variables:
     - **MODI method**: u_i (row) and v_j (column), set $u_1 = 0$, remaining with $c_{ij} = u_i + v_j$
       → improvement index: $I_{ij} = c_{ij} - u_i - v_j$
     - **Stepping-Stone method**: closed path (+ / –) → improvement index

     → EV = variable with largest negative reduced cost
   - Select the variable with the largest negative reduced cost to enter the basis.
3) Determine the leaving basic variable
   - **Pivoting**: closed path → + (recipient cells) and - (donor cells)

→ leaving variable = donor cell with smallest allocation

4) Determine the new basic feasible solution
   ⇨ Add the value of the leaving basic variable to the allocation for each recipient cell and subtracting this same amount from the allocation for each donor cell.

Sensitivity Analysis for Transportation Problems (exercise 6)

Changing the objective function coefficient c_{ij} of a **non-basic variable** (unoccupied cell) into c'_{ij}

- The current basic solution remains feasible (RHS is not changed)
- The shadow prices u_i and v_j remain unchanged
  ⇨ Range of values? Change c_{ij} to $c_{ij} + \Delta$.
- Current basic solution remains optimal if $c'_{ij} - u_i - v_j > 0$

Changing the objective function coefficient c_{ij} of a **basic variable** (occupied cell) into c'_{ij}

- The current basic solution remains feasible (RHS is not changed)
- Find *new shadow prices* u_i and v_j as the set of equations have changed
  ⇨ Range of values? Change c_{ij} to $c_{ij} + \Delta$ → than new shadow prices
- Current basic solution remains optimal as long as all *non-basic variables price out nonnegative*.

Increasing both supply s_i and demand d_j by Δ

- This change maintains a balanced transportation problem (Total supply = Total demand)
- If x_{ij} is a basic variable in the optimal solution then increase x_{ij} by Δ.
- If x_{ij} is a non-basic variable in the optimal solution then find the loop involving x_{ij} and some of the basic variables. Mark this cell as the leaving variable and trace a closed path back to this non-basic variable, alternately increasing and then decreasing the current basic variables in the loop by Δ.

The Assignment Problem

An **assignment problem** is a balanced transportation problem in which all supplies (assignees) and demands (tasks) are equal to 1.

Objective:

Minimize the total cost assignment.

Methods:

- The Transportation Simplex method
- The general purpose Simplex Method
- The Network Simplex Method

Considerations:

- If the number of assignees does not equal the number of tasks, dummy assignees or tasks need to be added with 0 assignment costs.
- If assignee i cannot perform task j, assign a large positive cost M to c_{ij}.
- To solve an assignment problem that maximizes the objective function, 2 methods:
  1) Create an opportunity loss matrix:
     → Subtract all profits for each task from the maximum profit for that task and solve the problem as a minimization problem.
  2) Transform the profits matrix into a cost matrix
     → Multiply the profits through by -1 and solve the problem as a minimization problem.

The Hungarian Method

- Reduced cost matrix
- Solution method: see slides!

The Transshipment Problem

Transshipment problems are transportation problems in which a shipment may move through intermediate nodes before reaching a particular destination node.
→ k **transshipment points** or **junctions** through which goods are shipped.

Chapter 3: Network Optimization Problems

The Shortest-Path Problem

Objective
Find the shortest path (the path with the minimum total distance) from the origin to the destination.

Dijkstra's labeling algorithm

$[l_x, k_x]$ → permanent label
(l_x, k_x) → tentative (= voorlopig) label.
With l_x = distance from the source node, k_x = preceding node

The Minimum Spanning Tree Problem

Objective
Choose the set of links that have the shortest total length among all sets of links that satisfy the condition that a path between every pair of nodes is provided.
Tree: (n-1) links & n nodes

Prim's algorithm

Cheapest arc between a marked node and an unmarked node.

Kruskal's algorithm

Cheapest arc between:

- Two unmarked nodes.
- A marked node and an unmarked node.
- Two marked nodes that are not connected.

The Maximum Flow Problem

Objective
Maximize the total amount of flow from the source to the sink.
Caution!
In case of multiple sources and/or multiple sinks, create a single **dummy source** out of which all flow originates and/or **dummy sink** that absorbs all the flow that terminates at the sinks.

Ford-Fulkerson algorithm

Step 1: Find any directed path from the source to the sink that has a strictly positive flow capacity remaining (= identify an **augmenting path**). **Stop** if no such path exists, the optimal solution is found.
Step 2: Determine f, the maximum flow along this path, given by the smallest flow capacity of all arcs in the path. Increase the flow through the network by sending the amount f over this path.
Step 3: Subtract f from the remaining flow capacity in the forward direction for each arc in the path, and add f to the remaining flow capacity in the backward direction for each arc in the path. Go to Step 1.

The Minimum Cut Problem

Objective
Find the cut that has the minimum cut value.

Solution method

Ford-Fulkerson algorithm

Max flow min cut theorem

For any network having a single origin node and a single destination node, the **maximum possible flow** from origin to destination **equals** the **minimum cut value** for all cuts in the network.

Weak duality property

Maximum flow value ≤ capacity of any cut

→ capacity of any cut is an *upper bound* on the maximum flow.

Strong Duality Property

Maximum flow value = Capacity of minimum cut

The Minimum Cost Flow Problem

Objective

Minimize the total cost of sending the available supply through the network to satisfy the demand.

Complex Network Optimization Problems

Travelling Salesman Problem (TSP)

Make a tour at minimal costs that visits all customers exactly once.

Intermezzo

BIP Encoding

Use Big M to turn-off constraint:

Either:

- $3x_1 + 2x_2 < 18$
- and $x_1 + 4x_2 < 16 + M$ (and M is very BIG)

Or:

- $3x_1 + 2x_2 < 18 + M$
- and $x_1 + 4x_2 < 16$

Use binary y to decide which constraint to turn off:

- $3x_1 + 2x_2 < 18 + y\,M$
- $x_1 + 2x_2 < 16 + (1-y)M$
- $y \in \{0,1\}$

LP Encoding

Introduce y_i to turn off each constraint i

Use Big M to turn-off constraint:

- $f_1(x_1, \dots, x_n) < d_1 + My_1$,
- $f_N(x_1, \dots, x_n) < d_N + My_N$

Constrain K of the y_i to select constraints:

- K out of N constraints hold: $\sum_{i=1}^{N} y_i = N - K$
- At least K of N hold: $\sum_{i=1}^{N} y_i \leq N - K$

Chapter 4: Integer Programming

Integer Programming

http://web.mit.edu/15.053/www/AMP-Chapter-09.pdf

Integrality requirement

One or more variables must assume an integer value.

Types of integer linear programming models

- Pure integer (linear) program (**PIP**) → All integer LP
  - ➢ A linear program in which all variables are required to be integers.
- Binary integer program (**BIP**)
  - ➢ All variables in an LP are restricted to be 0 or 1 → **binary variables**
- Mixed integer (linear) program (**MIP**)
  - ➢ Only a subset of the variables are restricted to be integer or binary.

Applicability of Integer Programming

The use of discrete variables

- Indivisible (discrete) quantities
- Yes-or-no (binary) decision variables
- **Indicator variables y_i**: binary variables that are linked to continuous variables to indicate certain states.

IP Formulation

Formulating logical constraints: see slides 25-33

- Contingent decisions: A implies B
- Pack constraint: at most one out of n items can be selected.
- Partition constraint: exactly one out of n items must be selected.
- Cover constraint: at least one out of n items must be selected.
- Cardinality constraint: more than k out of n items must be selected.
- K out of the N constraints must hold → indicator variables y_i whether or not constraint i is satisfied.
- Variable upper bound constraint: "If you build a warehouse, you can store up to 13 tons of x in it."
- Semi-continuous variable: "If you build a warehouse, it must be used to store at least 56 tons but no more than 141 tons."
- Binary factorization
- The **knapsack** problem
  - n items, with per unit values and weights v_i and w_i (i = 1, …, n), can be put into a bag that can carry a maximal weight of W.
  - How many of each item should be put into the bag such that the total value is maximized?
- The **fixed-charge** problem → a fixed or lump-sum cost
- Blending problem

Solution methods

Rounding the LP solution

LP relaxation (LPR):

- Ignoring the integrality constraints.
- LP solution
- Rounding up or down to integer values → may be infeasible or sub-optimal

➔ Complete enumeration of feasible ILP solutions

Enumeration procedure

Make sure that only a small fraction of the feasible solutions need to be examined:

- Dynamic programming (see later)
- Branch-and-bound: divide-and-conquer method
  - Initialisation: set $Z^* = -\infty$
  - Stopping rules:
    - Optimality test: stop when there are no remaining subproblems to investigate, the current incumbent is optimal.
    - Specify a sub-optimality tolerance factor: allows to stop once an integer solution is found that is within some % of the optimal value.
  - Branch-and-bound with binary variables:
    - Dividing (branching): partition the set of feasible solutions into smaller subsets.
      - **Branching**: partition into subsets by focusing on one variable.
    - Conquering (bounding + fathoming): find a bound (Z*) for how good the best solution in a subset can be and then discard the subset if its bound indicates that it can't contain an optimal solution.
      - **Bounding**: a bound for sub-problems is obtained by solving the LP relaxation. (Note that the optimum of an IP can never be better than the optimum of its LP relaxation.)
      - **Fathoming**: dismissing a subproblem from further consideration.
        - The LP relaxed solution is worse than the best known integer solution.
        - The LP relaxation is infeasible.
        - The LP relaxed solution is all integer (update best known integer solution).
  - Branch-and-bound with general integer variables:
    - **Branching**: select the first integer-restricted variable with a fractional value in the LP relaxed solution and create two new subproblems.
    - **Bounding**: calculate the optimum of the LP relaxation.
    - **Fathoming**: dismiss a subproblem if
      - The LP solution is worse than the best known integer solution.
      - The LP relaxation is infeasible.
      - The LP solution is integer.

Branch-and-cut for BIP

- Automatic BIP preprocessing
  - Fixing variables
    - If one value of a variable cannot satisfy a constraint (even when the other variables equal their best value for trying to satisfy the constraint), then that variable can be fixed at its other value.
  - Eliminating redundant constraints
    - If a functional constraint satisfies even the most challenging binary solution, then it has been made redundant by the binary constraints and can be eliminated.
  - Tightening $\leq$ constraints
    - A constraint can be tightened by adjusting the coefficients and right-hand side such that the feasible region of the LP relaxation reduces without eliminating feasible binary solutions.
- Generation of cutting planes
  - A **cutting plane** (or *cut* or *valid inequality*) = a new functional constraint that reduces the feasible region for the LP relaxation without eliminating feasible solutions for the IP problem.
  - **Minimum cover cuts** (for BIP) → for functional constraints in ≤ form with only nonnegative coefficients.
  - **Gomory cuts** (for PIP) → when all coefficients and right-hand sides have integer values.
    - Gomory cuts make the optimum of the LP relaxation infeasible by adding a constraint that cuts the LP optimum from the solution space, based on the fractional values that appear in the LP optimum.

Chapter 5: Dynamic Programming

General Description

General description

The problem can be divided in N separate **stages** with a policy decision required at each stage.
Each stage has a number of **states** associated with the beginning of that stage.

- In stage n the system has a *certain state* x_n.

The effect of the policy **decision** $\boldsymbol{d_n}$ at stage n is to transform the current state x_n to a state x_{n+1} associated with the beginning of the next stage.

- The *return function* $r_n(x_n, d_n)$ gives the 'return' of making decision d_n being in state x_n at stage n.
- The *state transition function* $t_n(x_n, d_n)$ gives the state x_{n+1} at stage n+1 that is reached after making decision d_n in state x_n at stage n.

Recursive relationship

The **cumulative return** is the sum of the immediate return in the current stage and the cumulative return in the remaining stages:

- $f_n(x_n, d_n) = r_n(x_n,d_n) + f^*_{n+1}(x_{n+1})$ with $x_{n+1} = t_n(x_n,d_n)$

The **optimal cumulative return** gives the best cumulative return in each state:

- $f^*_n(x_n) = max/min_{d_n}\ (f_n(x_n,d_n))$

In the final stage N, there is no next stage, so the cumulative return is equal to the immediate return:

- $f_N(x_N, d_N) = r_N(x_N, d_N)$
- $f^*_N(x_N) = max/min_{d_N}\ (r_N(x_N, d_N))$

Backward recursion: to solve the problem, start in stage N and work back to stage 1.

Table for exercises:

Chapter 6: Non Linear Programming

The general form of a nonlinear programming problem is to find x = (x_1 , x_2 , …, x_n) so as to:

- Maximize f(x)
- subject to: $g_i(x) \leq b_i$ for i = 1, 2, …, m
  x ≥0

where f(x) and the $g_i(x)$ are given functions of the n decision variables.

One-variable unconstrained optimization

The solution to the one-variable unconstrained NLP (maximize {f(x): x∈[a, b]}) may occur at one of the four following cases:

- Case 1: df(x) exists and reaches a value of 0 within the range [a, b].
  - **ANALTYICAL** procedures:
    - For a *concave* function f(x), x = x* is a *global maximum* if and only if df(x)/dx = 0 at x = x*.
    - For a *convex* function f(x), x = x* is a *global minimum* if and only if df(x)/dx = 0 at x = x*.
- Case 2: df(x) does not exist (i.e. at a discontinuity).
  - ⇨ Test points on either side of the discontinuity.
- Case 3: the endpoints a and b of the interval [a, b].
- Case 4: for nonlinear functions f(x), the derivative df(x)/dx may still be nonlinear and you may not be able to solve the equation analytically.

- **NUMERICAL** search procedures: see slides 29-34
  - Bisection method → $x' = \frac{\underline{x}+\bar{x}}{2}$ → STOP when $\bar{x} - \underline{x} \leq 2\varepsilon$

    Table:

  - Newton's method → $x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$ → STOP when $|x_{i+1} - x_i| \leq \varepsilon$

    Table:

Multivariable unconstrained optimization

The point x* = (x_1^*,x_2^*, … ,x_n^*) is called a **stationary point** if it satisfies: $\frac{\partial f(x^*)}{\partial x_i} = 0$ for all i = 1, …, m.

Gradient $\nabla f(x)$ = vector of all partial derivatives → defines direction in which rate increase of f(x) is maximal.

Gradient search

Step 1: Initialization

- Select ε and any trial solution x_0 .

Step 2:

- If all partial derivatives in the solution are smaller than ε, stop.

Step 3: We want to move as far as possible, say t* units, in this direction.

- Thus, select the next point $\boldsymbol{x_{m+1} = x_m + t^*\nabla f(x_m)}$.
- Moving in the direction of the gradient ∇f(x_m) increases the function value f.
- Therefore, express $\boldsymbol{f(x_m + t\,\nabla f(x_m))}$ as a function of t.

Step 4: How big can t* be?

- Use one-variable unconstrained optimization to find **t = t*** that **maximizes** $\boldsymbol{f(x_m + t\,\nabla f(x_m))}$ over t > 0.
- Note: If it is a minimization problem, we **minimize** $\boldsymbol{f(x_m + t^*\nabla f(x_m))}$ subject to the restriction that t* > 0.

Step 5:

- Update the solution: x' ≡ x' + t*∇f(x').

Step 6:

- Once we have found our new point $x_{m+1} = x_m + t^*\nabla f(x_m)$, we evaluate $||\nabla f(x_{m+1})||$. If the length of the gradient is less than 0.01, then we have found a local maximum and can stop. Otherwise, return to step 2.

Table:

Constrained optimization

Consider the following NLP problem type, in which the restrictions are equality constraints ($\mathrm{CNLP}^{=}$):

- Maximize (Minimize) z = f(x_1, x_2, …, x_n)
- Subject: $g_i(x_1, x_2, \ldots, x_n) = b_i$, i = 1,…,m

Lagrangian relaxation

= convert the constrained $\mathrm{NLP}^{=}$ into an unconstrained NLP and solve it as usual.

Lagrangian function:

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i [(b_i - g_i(x)]$$ with λ_i = Lagrangian multiplier for constraint i.

➔ **Lagrangian multiplier $\boldsymbol{\lambda_i}$:** rate of change in the optimal value of f(x) per unit increase in b_i.

Karush-Kuhn-Tucker conditions

x = (x_1, x_2, …, x_n) can be an optimal solution for the nonlinear programming problem (NLP)

Maximize f(x)
s.t. $g_i(x) \leq b_i$, for all i = 1, 2, …, m
$x \geq 0$.

if there exist m numbers u_1, u_2, …, u_m such that all the KKT conditions are satisfied.

If f(x) is *concave* and all $g_i(x)$ are *convex*, then a solution satisfying the KKT conditions is **optimal**.

1) $\frac{df'(x,\lambda)}{dx} \leq 0 \quad \forall x$
2) $x\frac{df'(x,\lambda)}{dx} = 0 \quad \forall x$
3) $g(x) \leq b \rightarrow g(x) - b \leq 0$
4) $\lambda\,(g(x) - b) = 0$
5) $x \geq 0$
6) $\lambda \geq 0$

Chapter 7: Decision Making Under Uncertainty

Decision Analysis

Decision making without experimentation

Decision making under uncertainty

- **Maximax** (minimin) criterion (optimistic approach)
  - Choose the alternative with the *largest maximum payoff*.
  - If the payoff table was in terms of costs, the decision with the lowest cost would be chosen.
- **Maximin** (minimax) criterion (conservative approach)
  - For each decision the minimum payoff is listed and then the decision corresponding to the maximum of these minimum payoffs is selected (choose the alternative with the *largest minimum payoff*).
  - If the payoff was in terms of costs, the maximum costs would be determined for each decision and then the decision corresponding to the minimum of these maximum costs is selected.
- **Equally likely** criterion (or **Laplace** criterion)
  - Choose the alternative with the *largest average payoff.* The average payoff for every alternative is calculated and the one with maximum average payoff is picked.
  - This criterion assumes that probabilities of occurrence for the states of nature are equal.
- **Criterion of realism** (Hurwicz' criterion or **weighted average**)
  - Select *coefficient of realism* α between 0 and 1, then choose the alternative with *the largest value of*
    $$\alpha(maximal\ payoff\ for\ the\ alternative) + (1-\alpha)(minimal\ payoff\ for\ the\ alternative)$$
  - When α is close to 1, the decision maker is more optimistic about future; and when α is close to 0, the decision maker is more pessimistic about future.
- **Minimax regret criterion** (or **Savage** regret criterion)
  - This criterion requires construction of the *opportunity loss table* which is done by calculating for each state of nature the difference between each payoff and the largest payoff for that state of nature.
  - The *maximum regret* for each possible decision is listed.
  - Choose the alternative with the *smallest maximum opportunity loss* (= the minimum of the maximum regrets).

Decision making under risk → prior probabilities

- **Maximum likelihood** criterion
  - Identify the *most likely state of nature* and choose the alternative with the *largest payoff* for this state of nature.
- **Bayes' rule** (Expected Value approach)
  - The *expected return for each decision* is calculated by summing the products of the payoff under each state of nature and the probability of the respective state of nature occurring.
  - Choose the alternative with the *maximum expected payoff or expected monetary value* (**EMV**):

$$EMV_i = \sum_j r_{ij} p_j$$

with p_j probability of the j^{th} state of nature,
r_{ij} *payoff* of i^{th} alternative under the j^{th} state of nature.

- **Expected regret** approach
  - Choose the alternative with the *smallest expected regret or opportunity loss* (**EOL**):

$$EOL_i = \sum_j g_{ij} p_j$$

with p_j probability of the j^{th} state of nature,
g_{ij} *regret* for the i^{th} alternative under the j^{th} state of nature.

- The decision with the largest EMV will also have the smallest EOL.

Expected value of perfect information

The **expected value of perfect information (EVPI)** is the *increase in expected payoff* that would result from knowing with certainty which state of nature will occur.
→ *upper bound on the expected value* of any sampling or survey information.

EVPI calculation:

1) Determine the optimal return (with PI) corresponding to each state of nature.
2) Compute the expected value of these optimal returns.
3) Subtract the EV of the optimal decision from this expected value (determined in step 2):

EVPI = EV with PI - maximum EV

Decision making with experimentation (sample information)

Bayes' theorem

Given:

- n possible states of nature (indexed by i)
- m possible test findings (indexed by j)
- **Prior** probabilities P(state = i), for all i.
- **Conditional** probabilities P(finding = j | state = i), for all i and j.

Outcome:

- **Posterior** probabilities P(state = i | finding = j), for all i and j.

Calculate:

$$P(S_i \mid F_j) = \frac{P(S_i \cap F_j)}{P(F_j)} = \frac{P(F_j \mid S_i) P(S_i)}{\sum_i P(F_j \mid S_i) P(S_i)}$$

Posterior probabilities calculation:

1) For each state i, multiply its prior probability by the conditional probability for the finding j (indicator). This gives the *joint probabilities* for states and findings.
2) Sum these joint probabilities over all states. This gives the *marginal probability for each finding*.
3) For each state i, divide its joint probability with finding j by the marginal probability for that finding. This gives the *posterior probability*.

Expected value of experimentation (sample information)

The *additional expected profit* possible through knowledge of the sample or survey information.

EVSI Calculation:

1) Determine the optimal decision and its expected return for the possible outcomes of the sample or survey using the posterior probabilities for the states of nature.
2) Compute the expected value of these optimal returns.
3) Subtract the EV of the optimal decision obtained without using the sample information from the amount determined in Step 2.

EVSI = EV with experimentation – EV without experimentation

Decision trees

A decision tree has two types of **nodes**:

- The *decision nodes* (square nodes) correspond to *decision alternatives*: a decision must be made.
- The *event nodes* (round nodes) correspond to the *states of nature*: a random event will occur.

A decision tree has two types of **branches**:

- Branches *leaving decision nodes* represent the *different decision alternatives*, from which one must be selected.
- Branches *leaving event nodes* represent the *different states of nature*, each occurring with a given probability.

Markov Chains

Introduction: Stochastic processes

A **stochastic process** is a series of random variables $\{X_t\}$, with index t running through a given set T. The values X_t can assume are called the *states* of the process. We assume that the number of possible states M is finite. If the next state X_{t+1} of the process depends only on the present state X_t and is independent of past history, the process is called a **Markov Chain**.
The process will move from a state to other states with known **transition probabilities**.

Markov chains

The evolution over time of a (discrete time) stochastic process is described by the *(one-step) transition probabilities:*

- $P(X_{t+1}= j \mid X_t= i, X_{t-1}= s_{t-1}, \ldots, X_0= s_0)$

A stochastic process has the **Markovian property** if the transition probabilities depend on the current state only:

- $P(X_{t+1}= j \mid X_t= i, X_{t-1}= s_{t-1}, \ldots, X_0= s_0) = P(X_{t+1}= j \mid X_t= i)$

A stochastic process is a **Markov chain** if it has the Markovian property.
If the transition probabilities do not change over time, they are **stationary**:

- $P(X_{t+1}= j \mid X_t= i) = P(X_1= j \mid X_0= i) = p_{ij}$

This implies that the *n-step transition probabilities* are also stationary:

- $P(X_{t+n}= j \mid X_t= i) = P(X_n= j \mid X_0= i) = p_{ij}^{(n)}$

Chapman-Kolmogorov equations

The n-step probabilities can be obtained recursively from the one-step transition probabilities (with P the transition matrix).
The n-step transition probability matrix P(n) is given by the n'th power of P: $P^{(n)} = P^{(m-k)} \times P^{(k)}$

Steady-State Probability $\boldsymbol{\pi_j}$ for each state j is the *long-run probability* that a process starting in any state will be found in state j.

Classification of states in a Markov chain

States i and j are said to **communicate** if they are accessible from one another.
A Markov chain is **irreducible** if all states communicate, i.e. if there is only one **class**.

Transient states

A state is **transient** if the process *may never return to it after visiting it*. In other words, state i is transient if there exists a state j that is accessible from i, but not vice versa.

Recurrent states

A state is **recurrent** if the process *will certainly return to it after visiting it*. So, a state is recurrent if and only if it is not transient.
A state i is **absorbing** if the process *will never leave to another state once it enters* state i. An absorbing state is a special case of a recurrent state (p_{ii} = 1).

Periodicity

The **period** of state i is the integer t (t > 1) such that $p_{ii}^{(n)}$= 0 for all values of n other than t, 2t, 3t, …
If there are two consecutive numbers s and s+1 such that the process can be in state i at times s and s+1, the state is called **aperiodic**.
Aperiodic, *recurrent* states are called **ergodic** states.

Long-run properties of Markov chains

The **steady-state probabilities $\boldsymbol{\pi_j}$** can be calculated from $\boldsymbol{\pi = \pi \times P}$ and $\boldsymbol{\pi_1 + \pi_2 + \ldots + \pi_M = 1}$, i.e. the steady-state probabilities do not change from transition to transition and they cumulate to 1.

$$\pi_0 = \pi_0 P_{00} + \pi_1 P_{10}$$
$$\pi_1 = \pi_0 P_{01} + \pi_1 P_{11}$$
$$\pi_0 + \pi_1 = 1$$

Expected average cost

The *(long-run)* expected average cost per unit time:

$$\lim_{n\to\infty} E\left[\frac{1}{n}\sum_{t=1}^{n} C(X_t)\right] = \sum_{j=0}^{M} \pi_j C(j)$$

Other Markov Chain Concepts

The **first passage time** is the number of transitions (number of steps) necessary in *going from state i to j for the first time.*

➔ Expected first passage time μ_{ij}:

$$\mu_{ij} = 1 + \sum_{k\neq j} p_{ik}\mu_{kj}$$

The **recurrence time** is the number of transitions (number of steps) necessary *to return to state i* when starting from state i.

➔ Expected recurrence time μ_{ii}:

$$\mu_{ii} = \frac{1}{\pi_i}$$

Absorbing states

If there is more than one absorbing state in a Markov chain, the p*robability of going to absorbing state k from initial state i* is the probability of absorption f_{ik}.

Absorption probabilities $\mathbf{f_{ik}}$ for state k:

$$f_{ik} = \sum_{j=0}^{M} p_{ij} f_{jk}$$

$$with\ f_{kk} = 1\ and\ f_{ik} = 0\ if\ state\ i\ is\ recurrent\ and\ i \neq k.$$

Transition Matrix with Submatrices

If a **Markov chain** has *both absorbing and non-absorbing states*, the states may be rearranged so that the transition matrix can be written as the following composition of **four submatrices: I, 0, R, and Q**:

- I = an identity matrix indicating one always *remains in an absorbing state* once it is reached.
- 0 = a zero matrix representing 0 probability of transitioning *from the absorbing states to the non-absorbing states*.
- R = the transition probabilities *from the non-absorbing states to the absorbing states*.
- Q = the transition probabilities *between the non-absorbing states*.

The **fundamental matrix**, **N**, is the inverse of the difference between the identity matrix and the Q matrix:

$$N = (I - Q)^{-1}$$

The $\boldsymbol{N \cdot R}$ **matrix**, the product of the fundamental matrix and the R matrix, gives the probabilities of eventually moving from each non-absorbing state to each absorbing state.

Main questions on Exam!

- LP analysis (sensitivity analysis)
- Network optimization problems (Transportation, Transshipment, Assignment, Shortest path, Minimum spanning tree, etc.)
- IP modelling