Course: MSc DS

Optimisation

Module: 2

Learning Objectives:

1. Grasp the Fundamentals of Optimisation
2. Master the Warehouse Problem
3. Navigate the Assignment Problem
4. Delve into the Knapsack Problem
5. Hands-on Practice with Tools
6. Application in Real-world Scenarios

Structure:

2.1 Introduction to Various Types of optimisation Problems

optimisation is the process of finding the best solution or set of solutions from a set of feasible solutions, given a certain criterion or set of criteria. These criteria, often termed objective functions, measure the quality or performance of a solution. The importance of optimisation lies in its pervasive applicability in diverse fields, ranging from engineering to economics, and its role in decision-making.

2.1.1 Classification of optimisation Problems

optimisation problems can be classified along multiple dimensions, each highlighting a specific characteristic:

- **Linear vs. Non-linear**:
  - Linear optimisation: The objective function and the constraints are linear in nature. Simplex and interior point methods are classic solution techniques. Linear optimisation problems are relatively easier to solve and provide global optimum solutions.
  - Non-linear optimisation: Involves either a non-linear

    objective function, non-linear constraints, or both. They can be much more complex and may have multiple local optima. Techniques like gradient descent and evolutionary algorithms are commonly used.

- **Continuous vs. Discrete**:
    - Continuous optimisation: The decision variables are continuous. These problems often arise in settings such as design and production.
    - Discrete optimisation: The decision variables take on discrete values. Integer programming and combinatorial optimisation problems fall into this category.
- **Deterministic vs. Stochastic**:
    - Deterministic optimisation: All the parameters in the problem are known with certainty. The solutions to these problems remain consistent under the same conditions.
    - Stochastic optimisation: The problem incorporates randomness, either in the objective function, the

constraints, or both. This kind of optimisation is vital when dealing with uncertainty, like in financial modelling or supply chain management.

- **Single-objective vs. Multi-objective**:
    - Single-objective optimisation: Focuses on optimising a single objective function. The goal is to find a single solution that maximises or minimises the objective.
    - Multi-objective optimisation: Aims to optimise two or more conflicting objectives simultaneously. The outcome is typically a set of Pareto-optimal solutions where no objective can be improved without deteriorating another.

2.1.2 Challenges in optimisation:

optimisation, especially in the realm of data science, is not without its challenges:

- **Complexity**: The high-dimensional nature of many real-world problems can make them computationally intractable.
- **Multiple Local Optima**: Especially prevalent in non-linear

problems, the presence of multiple optima can lead algorithms to converge to suboptimal solutions.

- **Noise and Uncertainty**: Real-world data can be noisy, making it challenging to distinguish between genuine patterns and random fluctuations.
- **Model Mismatch**: The mathematical model might not fully capture the intricacies of the real-world scenario, leading to subpar decisions.
- **Scalability**: As data grows, optimisation problems can become harder to solve both in terms of computational time and memory.
- **Evaluation**: Especially in multi-objective optimisation, how do you measure the quality of a solution or compare solutions?

2.2 The Warehouse Problem

The warehouse problem is a classic optimisation conundrum, which primarily concerns the effective placement and management of goods within a storage facility. The core aim is to minimise storage and retrieval costs while adhering to specific constraints like space, handling efficiency, and accessibility. As warehouses play a pivotal role in the supply chain and logistics operations of numerous industries, optimising their function can lead to significant cost savings and improved operational efficiency.

2.2.1 Historical Context and Relevance

- Historically, warehouse management was a predominantly manual process. The advent of the industrial revolution and the boom of mass production necessitated better storage solutions.
- With time, as trade and commerce grew, so did the complexity of supply chains. This led to the realisation that efficient warehouse management could lead to cost reductions and quicker deliveries.
- In the modern era, the relevance of the warehouse problem has only increased, especially with the advent of e-commerce. There's an ever-growing demand for quicker deliveries, and with companies promising same-day or next-day shipments, efficient warehousing is paramount.

2.2.2 Basic Problem Definition

The fundamental warehouse problem can be distilled as follows:

Given a finite amount of goods, each with specific dimensions and retrieval frequencies, and a limited warehouse space with designated areas for storage:

- Where should each item be placed to minimise the overall storage and retrieval cost?

Mathematical Formulation

To translate the warehouse problem into a mathematical framework, we consider the following:

Objective Function: The primary goal is to minimise costs associated with storage and retrieval operations. This can be represented as: $\sum \min I = 1ncixi$ Where:

- n is the total number of goods.
- ci is the cost of storing or retrieving item i.
- xi is the decision variable for item i (e.g., its location in the warehouse).

Constraints: Constraints can include:

- Space limitations: The sum of all goods' space should not exceed warehouse capacity.
- Specific storage requirements: Certain goods may need refrigeration or be kept away from others due to compatibility concerns.

Decision Variables: xi for each item i, which can represent a variety of decisions, such as the exact location, the shelf level, or the section of the warehouse where item i is placed.

Solution Approaches

Solving the warehouse problem can be approached in multiple ways:

Graphical Method:

- This is a visualisation technique, suitable for problems with two decision variables.
- The feasible region is identified on a graph, and the optimal solution is determined visually or by evaluating the objective function at each corner point.

Linear Programming:

- When the objective function and constraints are linear, linear programming can be employed.
- Software tools like CPLEX or the Simplex method can be utilised for larger problems.

Metaheuristics:

- For more complex, real-world instances where exact methods are computationally prohibitive, metaheuristic algorithms like Genetic Algorithms, Simulated Annealing, or Particle Swarm optimisation can be used.

2.3 The Assignment Problem

The Assignment Problem is a fundamental combinatorial optimisation problem that can be succinctly described as the task of assigning n jobs to n workers in such a way that the total cost of assignment is minimised. Every worker can perform only one job, and every job is undertaken by only one worker. Such problems are pervasive in operations and logistics sectors where resource allocation is essential.

Problem Overview

- The matrix representation of the problem consists of a cost matrix where every element Cij represents the cost of assigning job i to worker j.
- The problem might have variations where the number of jobs isn't equal to the number of workers.

2.3.1 Significance in Operations and Logistics

In the operations and logistics sectors, efficient resource allocation ensures that tasks are accomplished swiftly, reducing wastage of time and resources. Some key points of significance include:

- **Cost Efficiency**: Minimising the assignment costs directly translates to monetary savings.
- **Resource optimisation**: Ensures that resources, whether they be human resources or machinery, are used optimally.
- **Operational Efficiency**: A proper assignment can lead to smoother operations with fewer disruptions.

2.3.2 Mathematical Formulation

- **Objective Function**:

  - Minimise $\sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}x_{ij}$

Where x_{ij} is a binary variable, which is 1 if job i is assigned to worker j and 0 otherwise. C_{ij} is the cost of assigning job i to worker j.

- **Constraints**:
  - Every job is assigned to exactly one worker: $\sum_{j=1}^{n} x_{ij}=1$ for all i.
  - Every worker is assigned exactly one job: $\sum_{i=1}^{n} x_{ij}=1$ for all j.
  - $x_{ij} \in \{0,1\}$
- **Decision Variables**:
  - x_{ij}: A binary decision variable that is 1 if job i is assigned to worker j and 0 otherwise.

2.3.3 Solution Approaches

Hungarian Algorithm: This is a polynomial-time algorithm used specifically for the assignment problem. It is efficient and guarantees finding the optimal solution.

It involves modifying the cost matrix to produce a matrix with as many zero elements as possible, then determining the optimal assignment using this zero matrix.

Branch and Bound:

- A general algorithm for integer programming problems, but can be specialised for the assignment problem.
- It involves partitioning the problem into subsets, bounding the solution for each subset, and using these bounds to prune branches that cannot produce a better solution.

Variants and Hybrid Methods:

- There are various problem extensions, like the generalised assignment problem, where not all workers are equally efficient at all jobs. Solutions may need to be adapted or hybridised for these cases.
- Hybrid methods might combine elements of different algorithms for enhanced efficiency or to tackle specific problem variants.

2.4 The Knapsack Problem

The Knapsack Problem (KP) is a classic problem in combinatorial optimisation that has been studied for over a century. Throughout its history, the problem has captured the imagination of many researchers due to its simplicity in description, yet complexity in solution.

2.4.1 History and Importance

- **Origins**: The problem dates back to at least the early 19th century and has its origins in practical applications, such as resource allocation and cargo loading.
- **Significance**: It serves as a fundamental problem in combinatorial optimisation. The Knapsack Problem's importance extends beyond its practical applications. Its many variants and related problems make it a cornerstone in the study of algorithms and complexity. Furthermore, it is often used as a benchmark for testing new optimisation methods.

2.4.2 Problem Description

Given a set of items, each with a weight and a value, determine the number of each item to include in a knapsack so that the total weight does not exceed a given limit while maximising the total value.

Mathematical Formulation

- **Objective Function**: Maximise the total value of the items in the knapsack.

Maximise ∑i=1nvixi

where vi is the value of item i and xi is a binary decision variable that denotes whether item i is included (1) or not (0).

- **Constraints**: The total weight of the items in the knapsack should not exceed the capacity, W. ∑i=1nwixi≤W

where wi is the weight of item i.

- **Decision Variables**: xi∈{0,1}

for each item i, where xi is 1 if the item is included and 0 otherwise.

Solution Techniques

1. **Dynamic Programming**:
    - The dynamic programming solution, specifically the 0/1

Knapsack, breaks down the problem into smaller, more manageable sub-problems.

- The solution uses a 2D table where the entry at the ith row and jth column represents the maximum value obtainable with weight j using the first i items.
- It leverages the principle of optimality, ensuring that the optimal solution to a problem relies on the optimal solutions to its sub-problems.

2. **Greedy Algorithm**:
   - This technique builds the solution piece by piece, always choosing the next piece that offers the most immediate benefit.
   - For the Knapsack Problem, the common greedy criterion is to select items based on their value-to-weight ratio.
   - While this method can provide an efficient solution, it does not guarantee an optimal solution for all instances.

3. **Genetic Algorithms**:
   - Genetic algorithms (GAs) are search heuristics inspired by the process of natural selection.
   - GAs represent solutions as chromosomes and apply genetic operations like mutation, crossover, and selection to evolve and improve the solutions over generations.
   - They are particularly useful for large-scale instances where traditional methods are not feasible.
4. **Approximation Algorithms**:
   - Approximation algorithms aim to find a solution that is close to the optimal in polynomial time.
   - They provide a guarantee on how close the solution is to the optimal.
   - For the Knapsack Problem, a popular approximation algorithm involves rounding off fractional solutions of the problem's relaxation.

2.5 Hands-on: Formulating optimisation Problems in Excel and

Python

Formulation in optimisation refers to the process of translating a real-world problem into a mathematical representation, often expressed as an objective function with constraints. A practical formulation is essential for several reasons:

- Accuracy: A well-formulated problem will accurately mirror real-world conditions, leading to solutions that are truly applicable in practice.
- Efficiency: Proper formulation ensures that the problem can be solved using optimisation techniques within a reasonable timeframe.
- Flexibility: With a practical formulation, it becomes easier to adjust or modify the model to address variations or changes in the problem scenario.

2.5.1 Using Excel for optimisation

Microsoft Excel, with its widely-used interface and versatile functions, offers an excellent platform for optimisation. Excel isn't just a spreadsheet tool; it's a powerful computational tool suitable for many optimisation problems encountered in industry and academia. The benefits of using Excel include:

- Accessibility: Many professionals and students are familiar with Excel, making it an accessible tool for optimisation tasks.
- Solver Integration: Excel comes with an optimisation tool called Solver, which allows for the solution of linear and non-linear optimisation problems.
- Data Manipulation: Given Excel's primary functionality as a spreadsheet tool, it's adept at managing and analysing data, which is often a precursor to optimisation.

2.5.2 Setting up the Problem

Before diving into the optimisation process, it's imperative to set up the problem correctly:

- Identify the Objective: Determine what needs to be maximised or minimised. This could be profit, cost, time, etc.
- Determine the Decision Variables: These are the variables that can be controlled or decided upon.
- Set Constraints: These are the limitations or restrictions on

the decision variables. They could be budgetary constraints, resource constraints, etc.

- Formulate Mathematically: Convert the objective and constraints into mathematical equations or inequalities.

2.5.3 Excel Solver: Step-by-step Guide

The Excel Solver is a tool that aids in finding the optimal solution to a problem. Here's a basic guide to using it:

1. Set Up the Spreadsheet: Before using Solver, input all the data, variables, and formulate the objective function and constraints in Excel cells.
2. Access Solver: Click on the "Data" tab in Excel and then select "Solver" from the Analysis group.
3. Define Objective: Specify the cell that represents the objective function. Choose whether you want to maximise, minimise, or achieve a certain value.
4. Set Decision Variables: Specify the cells that contain the decision variables.
5. Specify Constraints: Click on the "Add" button and input the

constraints by referencing the appropriate cells.

6. Choose a Solving Method: For linear problems, you'd typically select the Simplex LP method. For nonlinear problems, choose GRG Nonlinear or Evolutionary based on the nature of the problem.
7. Solve: Click on the "Solve" button.
8. Save or Discard: After the solution is found, decide whether to keep the Solver solution or revert to the original values.

2.5.4 Interpreting Results and Sensitivity Analysis

After running Solver, it's crucial to understand the results:

- Solution Report: This provides details about the solution, like the objective function value, decision variable values, and constraints' status.
- Sensitivity Report: This is useful for linear problems. It offers insight into how sensitive the solution is to changes in coefficients, offering details on shadow prices and allowable increases/decreases.
- Limitations and Assumptions: Always review the assumptions

made during the formulation and recognize any limitations of the model.

2.5.5 Python for optimisation in Data Science

optimisation plays a quintessential role in improving algorithms, decision-making processes, and operational efficiencies. The Python programming language, given its flexibility and extensive libraries, serves as an invaluable tool for optimisation.

Essential Libraries for optimisation:

- **PuLP**: PuLP is an open-source linear programming (LP) modeller written in Python. It assists users in the creation of LP problems, enabling problem definition, solution, and post-solution analysis.

  Features:

  - Allows for the creation of decision variables with ease.
  - Supports a wide array of linear solvers, such as CBC, GLPK, and more.
  - Intuitive syntax to define objective functions and constraints.

- **SciPy**: A fundamental library for scientific computing, SciPy comes with modules for optimisation, linear algebra, integration, and more. Its optimisation module provides algorithms for function minimization (or maximisation), root finding, and curve fitting.

  Features:

  - Implements a range of optimisation algorithms like BFGS, Nelder-Mead simplex, Newton-Conjugate Gradient, etc.
  - Offers both unconstrained and constrained optimisation techniques.
  - Integrated with other SciPy modules for a holistic scientific computing experience.

- **Gurobi**: Gurobi is a commercial optimisation solver renowned for its performance in solving large-scale linear programming (LP), mixed-integer linear programming (MILP), and quadratic programming (QP) problems.

  Features:

  - Known for its speed and robustness, especially for complex, large-scale problems.

- Provides a Python API for easy integration and problem definition.
- Comes with advanced functionalities like tuning, distributed optimisation, and more.

2.5.6 Defining and Solving optimisation Problems:

- **Problem Definition**:
    - Break the problem into objectives, decision variables, and constraints.
    - Determine the type of problem – linear, non-linear, integer, etc.
    - Model the problem mathematically, capturing real-world nuances and considerations.
- **Solution Approach**:
    - Choose the appropriate solver based on the problem type and scale.
    - Input the defined problem into the chosen Python library, ensuring the formulation aligns with the library's

requirements.

  - Execute the solver, ensuring computational efficiency and convergence.

Visualisation and Analysis of Solutions:

- Visualisation is paramount, especially in comprehending high-dimensional problems or in communicating insights to non-technical stakeholders.

  Tools: Libraries like **matplotlib**, **seaborn**, and **plotly** can visualise the optimisation landscape, solution paths, and feasible regions.

  Methods: Contour plots, surface plots, or convergence plots offer insights into the behaviour and quality of solutions.

- **Solution Analysis**:
  - **Validity**: Ensure the solution meets all the constraints and is feasible.
  - **Sensitivity Analysis**: Understand how changes in parameters or constraints might affect the solution.
  - **Benchmarking**: Compare solutions with known

benchmarks or other methods to gauge performance and reliability.

2.6 Summary

- ❖ optimisation Problems refer to mathematical procedures used to find the best solution from a set of feasible solutions, with its classification varying by linearity, continuity, determinism, and the number of objectives.
- ❖ The Warehouse Problem deals with the optimal arrangement or location strategy of goods inside a warehouse to minimise costs or maximise efficiency. Its solution involves determining the right placement of items concerning constraints and specific objectives.
- ❖ The Assignment Problem to the task of assigning jobs to workers (or any other similar entities) in such a way as to minimise total cost or maximise total efficiency. Solutions often utilise algorithms like the Hungarian method.
- ❖ A combinatorial problem, it involves selecting a set of items with given weights and values to place in a knapsack of

limited capacity so as to maximise the total value without exceeding the weight limit.

- ❖ Both tools are crucial for formulating and solving optimisation problems. While Excel offers user-friendly tools like Solver for linear problems, Python provides powerful libraries to tackle a broad range of optimisation issues.
- ❖ Beyond theoretical knowledge, it's essential to know how to formulate real-world optimisation problems and interpret results, which can be achieved using tools like Excel and Python for diverse applications.

2.7 Keywords

- **optimisation**: optimisation refers to the process of making something as effective or functional as possible. In the context of data science, it often pertains to finding the best solution from a set of possible solutions by minimising or maximising certain criteria, such as cost or profit.
- **Linear Programming (LP)**: Linear programming is a method used for achieving the best outcome (such as maximum profit

or lowest cost) in a mathematical model whose requirements are represented by linear relationships. It's widely used for optimisation when the problem can be expressed as linear equations and inequalities.

- **Metaheuristics** Metaheuristics are higher-level procedures designed to find, generate, or select a heuristic that might provide a sufficiently good solution to an optimisation problem, especially with incomplete or imperfect information. Examples include genetic algorithms and simulated annealing.
- **Hungarian Algorithm**: A combinatorial optimisation algorithm that solves the assignment problem in polynomial time. The assignment problem involves assigning tasks to agents such that the overall cost or time of task completion is minimised (or profit is maximised).
- **Dynamic Programming**: Dynamic programming is a method for solving problems by breaking them down into smaller subproblems. It's particularly useful for optimisation

problems like the knapsack problem. It avoids the computation of the same subproblems by storing their solutions in a table (typically implemented as an array or some sort of map), thus saving computation time at the expense of a (sometimes significant) space overhead.

- **PuLP**: PuLP is a Python library used for linear optimisation problems. With PuLP, users can create mathematical models, define constraints, and set objectives. After defining the problem, PuLP interfaces with various solvers (like CBC and GLPK) to find a solution.

2.8 Self-Assessment Questions

1. How would you distinguish between deterministic and stochastic optimisation problems? Provide an example for each.
2. What are the primary differences between the Warehouse Problem, the Assignment Problem, and the Knapsack Problem in terms of objective functions and constraints?
3. Which solution approach is specifically known for solving the

Assignment Problem and how does it work?

4. What are the key steps to formulate an optimisation problem in Python using the PuLP library?
5. Which tool, Excel or Python, would you choose for solving a large-scale optimisation problem and why?

2.9 Case Study

Title: optimisation of Delivery Routes for Swiggy in Mumbai

Introduction:

Swiggy, one of India's most popular food delivery services, faced an increasing challenge in the bustling city of Mumbai. With the city’s intricate layout, frequent traffic jams, and evolving urban landscape, ensuring timely delivery of hot meals became a significant hurdle. The brand’s promise of quick delivery was at stake, affecting its reputation and customer satisfaction.

In 2020, Swiggy's Mumbai operations team observed that although they had a large fleet of delivery personnel, there were still significant delays in many deliveries. A deep dive into the data revealed that the primary issue wasn't the number of delivery personnel but the efficiency of the routes they were taking. Traditional methods of choosing routes based on the driver's intuition or straightforward point-to-point GPS suggestions were not sufficient given Mumbai’s unique challenges.

Background:

Swiggy decided to employ advanced optimisation techniques to enhance their delivery routes. Collaborating with a team of data scientists, they developed an algorithm that factored in real-time traffic data, historical delivery times, road closures, and even localised events that could affect traffic. This approach utilised a combination of linear programming and heuristic methods to generate the most efficient routes.

Post-implementation, Swiggy observed a 15% reduction in average delivery times and a 10% decrease in fuel costs across their Mumbai fleet in just the first quarter. The optimised routes meant that their delivery partners were less stressed, leading to a drop in attrition rates. The most significant win, however, was the jump in customer satisfaction scores, ensuring Swiggy's stronghold in the competitive Mumbai market.

Questions:

1. What was the primary challenge faced by Swiggy's Mumbai operations team concerning deliveries?
2. How did the data scientists at Swiggy approach the problem of inefficient delivery routes? Mention the techniques used.
3. What were the key outcomes and benefits observed by Swiggy after implementing the optimised routes?

2.10 References

1. "Introduction to Operations Research" by Frederick S. Hillier and Gerald J. Lieberman
2. "Convex optimisation" by Stephen Boyd and Lieven Vandenberghe
3. "Linear Programming and Network Flows" by Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali
4. "Optimisation by Vector Space Methods" by David G. Luenberger
5. "The Art of Computer Programming, Volume 1: Fundamental Algorithms" by Donald E. Knuth.