Technical University of Košice

Faculty of Mining, Ecology, Process Control and Geotechnologies

Institute of Control and Informatization of Production Processes

Mathematical method for identification, modelling and simulation

Ing. Ales Jandera

11. July 2024

Contents

1 Advanced mathematical methods for decision and planning

Advanced mathematical methods for decision-making and planning are crucial in various fields, including operations research, economics, engineering, and artificial intelligence. These methods involve sophisticated mathematical techniques to analyze, model, and solve complex problems. Here are some key methods:

1.1 Linear Programming (LP)

Linear Programming involves optimizing a linear objective function subject to linear equality and inequality constraints [7]. The objective function represents a goal such as maximizing profit or minimizing cost. The constraints represent limitations or requirements, such as resource capacities, production capabilities, or budget limits. By mathematically modeling these objectives and constraints, linear programming helps identify the best possible outcome within the feasible region defined by the constraints. It's widely used in resource allocation, production planning, and logistics, providing valuable insights into how to efficiently distribute limited resources, schedule production runs, and manage supply chains. Additionally, linear programming can be applied to areas like transportation, where it aids in determining the most cost-effective routes for delivery, and in finance, where it helps in portfolio optimization to maximize returns or minimize risk.

Example of using the method:
Maximize

$$Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

Subject to:

$$\begin{gathered} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \end{gathered}$$

$$x_i \geq 0 \text{ for all } i$$

1.2 Dynamic Programming (DP)

Dynamic Programming solves complex problems by breaking them down into simpler subproblems, addressing each subproblem only once and storing its solution [1]. This approach avoids the redundant computations typical of naive recursive methods. Dynamic Programming is particularly useful for problems with overlapping subproblems, where the same subproblems arise multiple times, and optimal substructure properties, where the optimal solution to the problem can be constructed from optimal solutions to its subproblems. Applications of Dynamic Programming span various fields, including computer science for algorithm design, operations research for resource management, economics for decision-making, and bioinformatics for sequence alignment. Classic examples include the Fibonacci sequence, shortest path problems like Dijkstra's algorithm, knapsack problems, and dynamic programming on trees. By structuring these problems into a systematic framework, Dynamic Programming provides efficient and scalable solutions, making it a powerful tool in both theoretical and applied contexts.

Example of using the method:

Fibonacci sequence calculation:

$$F(n) = F(n-1) + F(n-2)$$

With base cases:

$$F(0) = 0, F(1) = 1$$

1.3 Markov Decision Processes (MDPs)

Decision-making in situations where outcomes are partly random and partly under the control of a decision maker [5]. An MDP is defined by its states, actions, transition probabilities, and rewards. The decision maker, or agent, chooses actions based on the current state, and transitions to a new state according to the transition probabilities, receiving a reward associated with the action taken. The goal is to find a policy, a mapping from states to actions,

that maximizes the expected cumulative reward over time. MDPs are widely used in robotics for path planning and control, in economics for modeling decision-making under uncertainty, and in artificial intelligence, particularly in reinforcement learning, where agents learn optimal policies through interactions with the environment. The framework of MDPs allows for the formalization and solution of a wide range of sequential decision-making problems, making them a fundamental tool in both theoretical research and practical applications.

Consists of:

- States S
- Actions A
- Transition probabilities $P(s'|s, a)$
- Rewards $R(s, a)$
- Policy $\pi(s)$

1.4 Game Theory

Game Theory studies strategic interactions between decision-makers, where the outcome for each participant or player depends on the actions of all involved [4]. It provides a framework for understanding and analyzing situations where players make decisions that are interdependent, leading to outcomes that depend on the choices of others. Game theory covers a variety of games, including cooperative and non-cooperative games, zero-sum and non-zero-sum games, and games of complete and incomplete information. It is used in economics to model market behavior, in political science to analyze voting systems and strategy, and in evolutionary biology to study the evolution of cooperation and competition among species. Game theory also finds applications in areas like computer science for algorithm design, network security, and artificial intelligence, particularly in multi-agent systems. Concepts such as Nash equilibrium, dominant strategies, and Pareto efficiency are central to game theory, providing insights into how rational players would behave in strategic situations, and guiding the design of mechanisms and systems for optimal decision-making.

Example:
Prisoner's Dilemma, where two players can either cooperate or defect. Both cooperate: each gets 3 points. Both defect: each gets 1 point. One cooperates, the other defects: cooperator gets 0, defector gets 5.

1.5 Bayesian Decision Theory

Bayesian Decision Theory incorporates probabilistic models to make decisions, providing a rigorous framework for statistical decision-making and machine learning [2]. It leverages Bayes' theorem to update the probability estimates of outcomes based on new evidence. The key components of Bayesian Decision Theory include prior probabilities, which represent initial beliefs about the likelihood of outcomes; likelihoods, which describe the probability of observing the evidence given an outcome; and posterior probabilities, which are updated beliefs after considering the evidence. Decision rules are then used to select the action that maximizes expected utility or minimizes expected loss based on these posterior probabilities. This theory is foundational in many areas, including medical diagnosis, where it helps in assessing the probability of diseases given symptoms; finance, for updating investment strategies based on market data; and machine learning, particularly in Bayesian inference methods, where models are continuously refined as more data becomes available. By incorporating uncertainty and learning from data, Bayesian Decision Theory provides a powerful approach to making informed and rational decisions in uncertain environments. Components: Prior probabilities, Likelihoods, Posterior probabilities, Decision rules.

1.6 Reinforcement Learning (RL)

Reinforcement Learning involves agents learning to make decisions by interacting with an environment [6]. The agent takes actions in the environment and receives feedback in the form of rewards or penalties. The goal of the agent is to learn a policy, which is a mapping from states to actions, that maximizes the cumulative reward over time. Reinforcement Learning is a key area in artificial intelligence and robotics, where it is used to develop systems that can adapt and improve their performance through experience. This approach is based on the principles of trial and error, and involves exploring the environment to discover which actions yield the highest rewards. Key con-

cepts in reinforcement learning include the value function, which estimates the expected reward for each state; the Q-function, which estimates the expected reward for state-action pairs; and algorithms such as Q-learning, policy gradients, and deep reinforcement learning, which combine reinforcement learning with deep neural networks. Applications of reinforcement learning span various domains, including autonomous vehicles, game playing (e.g., AlphaGo), robotic control, and personalized recommendations, showcasing its versatility and power in solving complex decision-making problems.

Q-Learning algorithm:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

2 Advanced methods in Graph Theory

Graph theory is a fundamental field of mathematics and computer science that studies the properties and applications of graphs, which consist of vertices connected by edges. It is essential for solving problems in various domains like **telecommunications** used for optimizing the routing of data to maximize throughput. **Transportation and Supply Chain Management** determining the most efficient way to route goods and services. **Utility Networks** to managing water, gas, or electricity distribution to maximize efficiency and reliability. **Project Management** using graph theory to task scheduling where tasks are dependent on the completion of preceding tasks. **Epidemiology** modeling the spread of diseases through dynamically changing human contact networks.

2.1 Network flow

Network flow is a fundamental concept in graph theory that models the flow of quantities through a network, allowing for the analysis and optimization of systems.

2.1.1 Basic Components of Network Flow

For our purpose as in operations research, we will use terminologyu as a directed graph is called a network, the vertices are called nodes and the edges are called arcs. **Directed Graph** used for a network flow problem

where each edge has a capacity and each node is either a source, a sink, or an intermediate node. Nodes represent junction points, and edges represent paths that the flow can take. Source are the origin node from where the flow starts and the sink is the destination node [3] where the flow is intended to end. Each edge in the network has an associated capacity, which is the maximum flow that the edge can handle.

Flow is the amount of stuff (e.g., data, goods, traffic) that passes along the edges. It must satisfy **Capacity Constraint condition** that flow on an edge cannot exceed the capacity of the edge and **Conservation of Flow condition** except for the source and sink, the flow into any node must equal the flow out of that node.

2.1.2 Key Concepts

Maximum Flow is the main problem in network flow to determine the maximum flow from the source to the sink without exceeding the capacities of the edges. **Flow Value** is the total amount of flow that leaves the source or enters the sink (these two values are equal due to the conservation of flow).

Residual Network is constructed from the original flow network by including residual capacities which indicate additional possible flow on an edge given the current flow. **Augmenting Path** in the residual network is a path from the source to the sink along which additional flow can be pushed to increase the overall flow in the network.

Cut in a network is a partition of the vertices into two disjoint subsets that separate the source from the sink. The capacity of the cut is the sum of the capacities of the edges that are directed from the subset containing the source to the subset containing the sink. The max-flow min-cut theorem states that the maximum value of an source to sink flow is equal to the minimum capacity of an same cut in the network.

2.2 Graph Coloring

Graph coloring is a fundamental concept in graph theory, which involves assigning colors to elements of a graph under certain constraints. The most common form of graph coloring is vertex coloring, where colors are assigned to vertices such that no two adjacent vertices has the same color. This concept is not only pivotal in theoretical mathematics but also has practical

applications in areas such as scheduling, networking, and the allocation of resources.

2.2.1 Types of Graph Coloring

The minimum number of colors needed to achieve **Vertex Coloring** is known as the graph's chromatic number.
Proper vertex coloring is a concept in graph theory where the vertices of a graph are colored in such a way that no two adjacent vertices share the same color. For a proper vertex coloring using k colors in a graph G with vertex set V, the coloring function c can be described as:

$$c : V \to \{1, 2, ..., k\} \tag{1}$$

Such that for every edge $(u, v) \in E$ (the edge set of G):

$$c(u) \neq c(v) \tag{2}$$

Edge Coloring incident similar to vertex coloring but applies to edges. Here, no two edges sharing a common vertex, can have the same color. The minimum number of colors needed for an edge coloring of a graph is called the chromatic index $\chi'(G)$. For a graph $G = (V, E)$ with vertex set V and edge set E, and a given number of colors k, an edge coloring is a function:

$$c : E \to \{1, 2, \ldots, k\} \tag{3}$$

Such that for any two edges e_1 and e_2 that share a common vertex, the following must hold:

$$c(e_1) \neq c(e_2) \tag{4}$$

Face Coloring is used in planar graphs where faces (the regions bounded by edges) must be colored in such a way that no two faces sharing a boundary segment have the same color. This is directly related to the Four Color Theorem 2.2.2, which states that four colors are sufficient to color any planar graph.

2.2.2 Key Concepts and Theorems

Four Color Theorem is one of the most famous theorems in graph theory, it states that four colors are sufficient to color the vertices of any planar

graph so that no two adjacent vertices use the same color. A planar graph is a graph that can be embedded in the plane, meaning it can be drawn on a flat surface without any of its edges crossing each other For any planar graph G can be phrased as:

$$\chi(G) \leq 4 \tag{5}$$

Brooks' Theorem provides a bound on the chromatic number of a graph. It states that any connected graph other than a complete graph or an odd cycle has chromatic number at most Δ (the maximum degree of the graph).

Hall's Marriage Theorem often used in edge coloring and matching problems, it provides a condition that guarantees the existence of a perfect matching in bipartite graphs, which is directly related to the graph coloring problem 2.2.1. It states that a bipartite graph has a perfect matching if and only if for every subset S of one part of the graph, the number of neighbors of S (denoted as $N(S)$) is at least as large as S:

$$|N(S)| \geq |S| \tag{6}$$

2.2.3 Algorithms for Graph Coloring

Greedy Coloring is a straightforward and easy-to-implement method where vertices are colored one at a time, using the smallest available color that hasn't been used by adjacent vertices. This method does not generally yield the minimum number of colors needed. The greedy algorithm attempts to color each vertex with the lowest number k where k is the smallest positive integer not used by its adjacent vertices. If $d(v)$ is the degree of vertex v, then a vertex can be colored with one of $d(v) + 1$ colors:

$$c(v) = \min\{k \in \mathbb{N} \mid k \notin \{c(u) \mid u \text{ is adjacent to } v\}\} \tag{7}$$

Backtracking Algorithm is a more exhaustive approach that can find the chromatic number of a graph but may be computationally expensive for large graphs. Formally, it's defined by:

$$\chi(G) = \min\{k \mid G \text{ is k-colorable}\} \tag{8}$$

Heuristic Algorithms solve many practical problems use heuristic or approximation algorithms to find a "good-enough" coloring. Examples include

the DSATUR algorithm, which orders vertices based on their degree and saturation.

2.3 Dynamic Graphs

Dynamic graphs are an extension of traditional graph theory that deals with graphs in which the structure changes over time. These changes can include the adding or removing of vertices and edges, as well as changes in the weights or labels of existing vertices and edges. Dynamic graphs are particularly relevant in fields where the basic relationships between entities are not static but evolve, such as social networks, traffic networks, telecommunications, and biological networks.

2.3.1 Key Concepts

Temporal Dimension use time as a fundamental component, allowing for the exploration of how graph properties evolve. This temporal dimension distinguishes dynamic graphs from static graphs, where the structure is fixed.

Look at three **Types of Changes** in dynamic graphs. **Vertex Dynamics** includes the adding or removing of nodes. New users joining a social network or stations being added to a transport network. **Edge Dynamics** involves the adding or removing of edges, which could represent forming or dissolving relationships in a social network or route changes in transport system. **Attribute Changes** the properties and weights of vertices and edges, such as changing bandwidth in communication networks or fluctuating relationship strengths in social networks.
Discrete Time Models changes are modeled at specific time steps or intervals in oposite Continuous Time Models evolves the graph continuously over time, and changes can occur at any moment.

2.3.2 Analytical Challenges

The dynamic nature adds a layer of **complexity** to traditional graph problems, making them computationally harder to solve.
Keeping track of changes over time requires **efficient memory management** and computational strategies, especially for large-scale networks.
Predicting how a graph will evolve involves understanding past trends and potentially complex dependencies between changes.

2.3.3 Problems and Algorithms in Dynamic Graphs

Dynamic Connectivity determines whether two vertices are connected by a path and how this connectivity changes over time. Efficient algorithms are required to update connectivity information as the graph evolves.

Dynamic Shortest Paths is try to find the shortest paths in a graph that changes over time is crucial for applications like dynamic routing in transportation and communication networks. The problem of maintaining shortest paths in a dynamic graph, where edges can be added, removed, or have their weights changed, can be described as follows:

Distance Update After Edge Weight Change:

$$d(u, v) = \min(d(u, v), d(u, k) + w(k, v)), \tag{9}$$

where, $d(u, v)$ represents the shortest path from u to v, and $w(k, v)$ represents the new weight of an edge that has been updated.

Community Detection in social and biological networks, identifying groups or communities that change over time can provide insights into the underlying structure and dynamics of the network. The modularity Q at any time t can be formulated as:

$$Q_t = \sum_{c \in C} \left[\frac{L_c}{L} - \left(\frac{d_c}{2L} \right)^2 \right], \tag{10}$$

where L_c is the number of edges within community c, d_c is the sum of degrees of the nodes in c, and L is the total number of edges in the graph.

Network Resilience analyzing how structural changes affect the stability and resilience of networks against failures or attacks.
Assessing network resilience in dynamic graphs might involve calculating the algebraic connectivity, represented by the Fiedler value (the second smallest eigenvalue of the Laplacian matrix), which can be influenced by dynamic changes:

$$L = D - A, \tag{11}$$

where D is the degree matrix and A is the adjacency matrix of the graph.
Algebraic Connectivity (Fiedler Value):

$$\lambda_2 = \min_{x \neq 0} \frac{x^T L x}{x^T x} \tag{12}$$

References

[1] Richard Bellman. *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[2] James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. New York, NY: Springer, 1985.

[3] Reinhard Diestel. *Graph Theory*. 5th. Berlin, Germany: Springer, 2017.

[4] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. Cambridge, MA: MIT Press, 1994.

[5] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ: John Wiley & Sons, 1994.

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. Cambridge, MA: MIT Press, 2018.

[7] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. 5th. New York, NY: Springer, 2020.