

LE07: Combinatorial optimization

Many optimization problems of great economic importance are so complex that an exact solution is often impossible. These problems typically belong to the class of **NP-hard problems** and are characterized by the fact that their solution requires **exponential** effort.

To solve practical problems, algorithms with **polynomial** runtime ($O(n^d)$) are needed, since algorithms with exponential runtime ($O(c^n)$) are mostly **infeasible**.

Complexity classes

A **decision problem** requires a “yes” or “no” answer. The class P consists of all decision problems that are solvable in polynomial time. The class NP encompasses decision problems for which a proof of the affirmative answer is **verifiable in polynomial time**. Obviously, $P \subseteq NP$.

A decision problem D is called **NP-complete** if $D \in NP$ and for every problem $D' \in NP$, $D' \propto_T D$. If such a problem D were in the class P , then this would imply: $P = NP$.

The transformation from D to D' , in which a polynomial algorithm transforms each instance I of D into an instance I' of D' such that the **answers** match, is called a **polynomial** transformation.

An example of an NP -complete problem is the **satisfiability problem**, which was first proven by Stephen A. Cook.

Graph problems and optimization

A subset $U \subseteq V$ in a graph $G = (V, E)$ is called a **clique** if $uv \in E$ for all $u, v \in U$. The problem of finding a clique with maximum cardinality is an optimization problem. **NP-complete** decision problems are the analogue of NP -hard optimization problems.

In combinatorial optimization, the finite set whose elements are called **feasible solutions** in practitioner jargon as **the search-space**. The optimization variant of finding a maximal clique in a graph is an **NP-hard problem**. **Maximization** problems are equivalent, since maximizing an objective function f is equivalent to **minimizing** $-f$.

Solution methods

A **backtracking algorithm** is a recursive procedure that generates all solutions to a combinatorial optimization problem step by step. An example is the *knapsack-backtrack1* algorithm for the knapsack problem. The knapsack problem is **NP-hard** and asks for a knapsack with maximum value whose size does not exceed the capacity K_0 .

To improve backtracking algorithms, the technique of **branch-and-bound** is used. This is possible if the root of the subtree (the partial solution) is not **feasible**, since then all partial solutions in that subtree are also not feasible.