ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering
and Networks Laboratory

Bio-inspired Optimization and Design

Eckart Zitzler

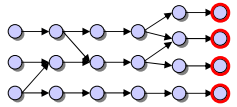
1. Optimization and Search

Optimization and Search in a Nutshell

specification = defining the goal of the decision maker (possibly informal)

modeling = formalizing the goal in terms of an optimization problem

optimization = searching within an ordered set of alternative solutions



In the Following...

...you learn:

- what is understood under the term optimization problem;
- what makes optimization problems difficult;
- what types of optimization methods can be distinguished;
- how in general complex optimization problems can be approached.

Introductory Example: The Coffee Machine Problem

Which combination of select buttons (switched *on*) generates something closest to latte macchiato?

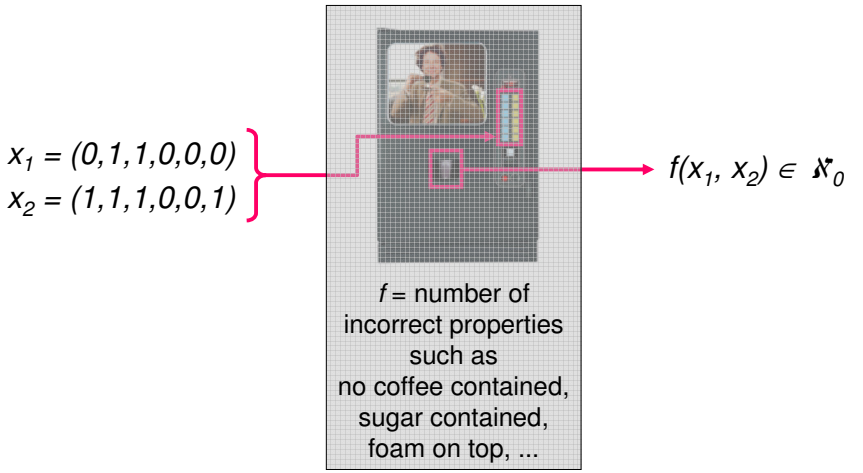
select buttons (*on* or *off*)

process button

Question

What strategy do you suggest to minimize the number of trials?

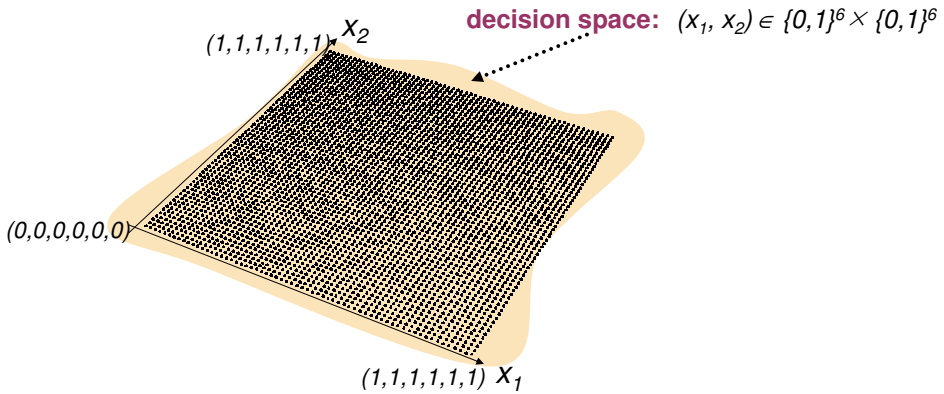
A Model for the Coffee Machine Problem



optimization problem: $((\{0,1\}^6, \{0,1\}^6), \aleph_0, f, \leq)$

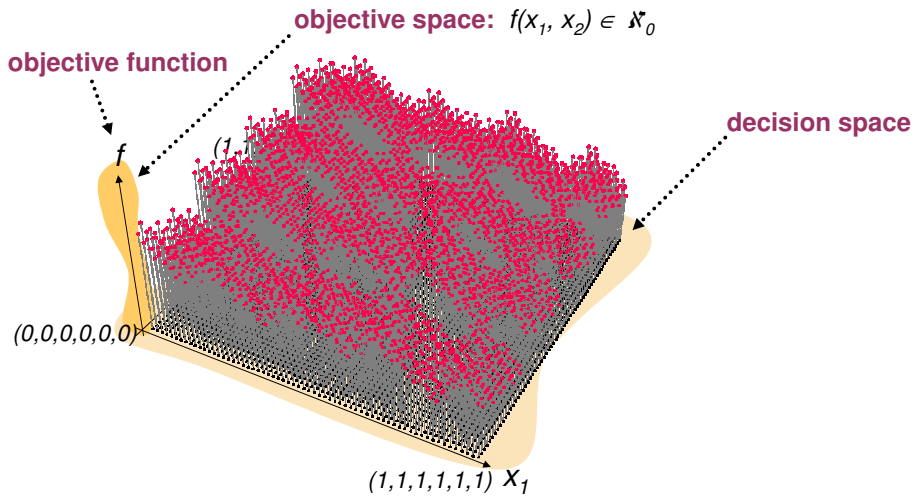
Coffee Machine Problem: The Decision Space

decision space = set of potential solutions to the problem



Coffee Machine Problem: The Objective Space

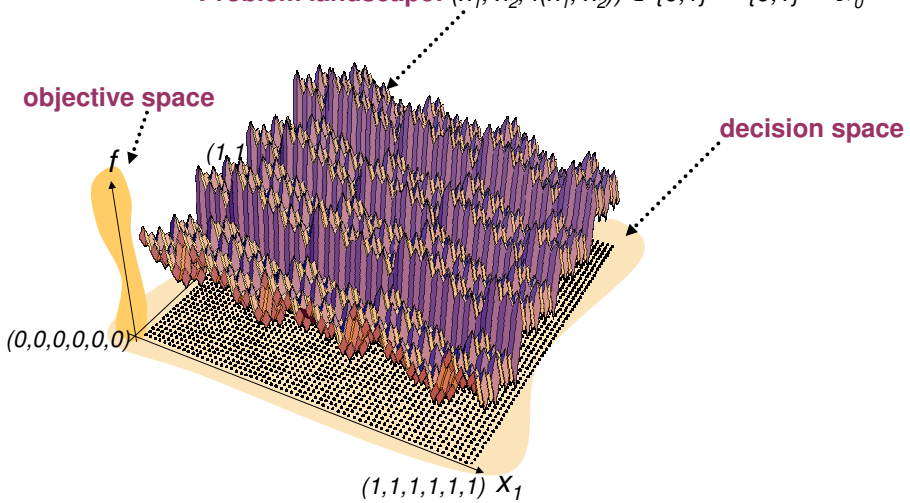
Objective space = space where solutions are compared to each other



The Coffee Machine Problem: Problem Landscape

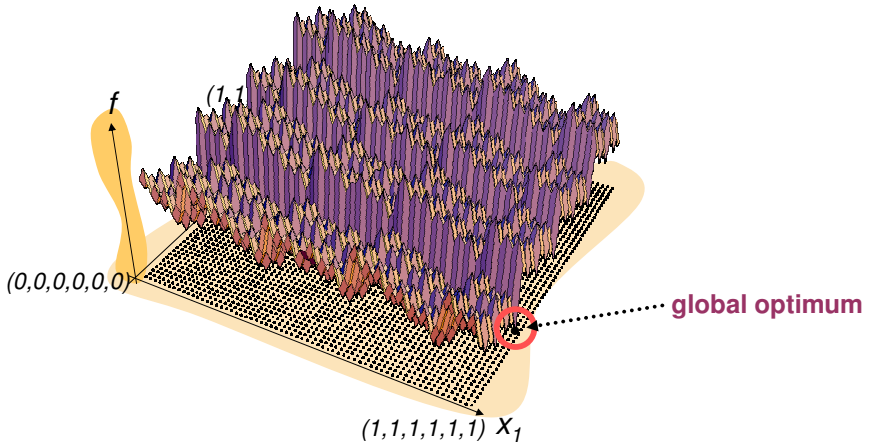
Problem landscape = search space and objective space together

Problem landscape: $(x_1, x_2, f(x_1, x_2)) \in \{0,1\}^6 \times \{0,1\}^6 \times \mathbb{N}_0$



The Coffee Machine Problem: Searching

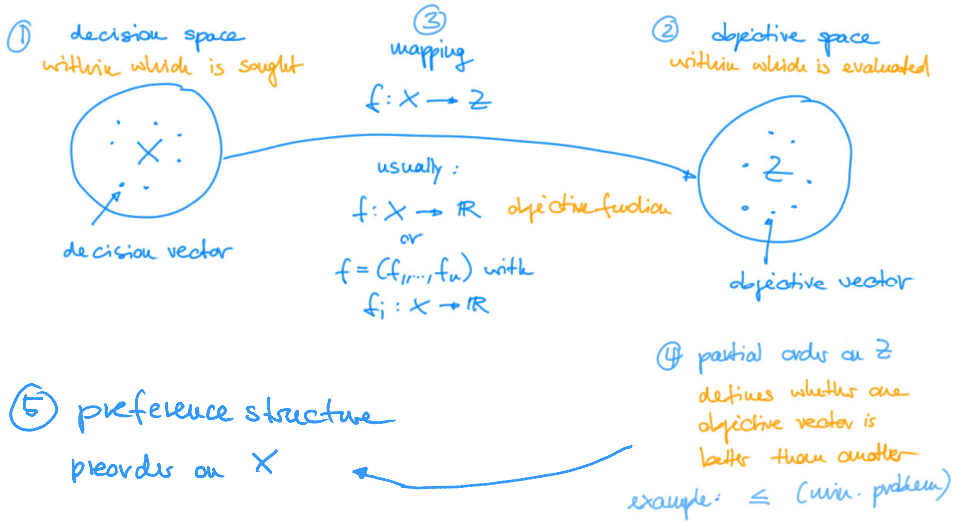
Optimization goal = identify an optimal solution



Intermediate Summary

- So far, we have seen how we to transform an informal specification („how to minimize the number of trials to obtain Latte Macciato?“) into a mathematical model.
- Next, we will generalize this and formally define what an optimization problem is.
- The following discussion may appear a little bit dry, but it is important to be precise here. In practice, one often observes that implementations have to be completely redesigned because the modeling part was done too sloppy.

What Is An Optimization Problem: Illustration



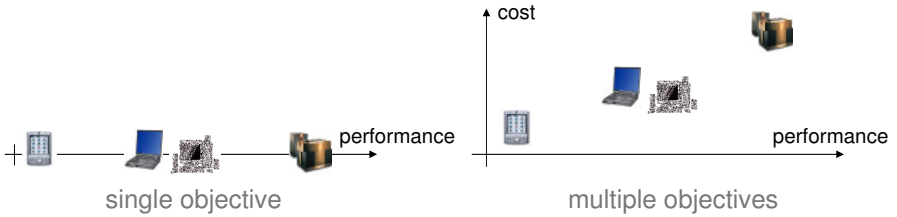
Optimization Problem: Definition

A general optimization problem is given by a quadruple *(X, Z, f, rel)* where

- *X* denotes the decision space containing the elements among which the best is sought; elements of *X* are called decision vectors or simply solutions;
- *Z* denotes the objective space, the space within which the decision vectors are evaluated and compared to each other; elements of *Z* are denoted as objective vectors;
- *f* represents a function $f: X \rightarrow Z$ that assigns each decision vector a corresponding objective vector; *f* is usually neither injective nor surjective;
- *rel* is a binary relation over *Z*, i.e., $rel \subseteq Z \times Z$, which represents a partial order over *Z*.

Objective Functions

- Usually, *f* consists of one or several functions $f_1, ..., f_n$ that assign each solution a real number. Such a function $f_i: X \rightarrow \Re$ is called an objective function, and examples are cost, size, execution time, etc.
- In the case of a single objective function (n=1), the problem is denoted as a single-objective optimization problem; a multiobjective optimization problem involves several (n ≥ 2) objective functions:



- In the following, we focus on single-objective optimization problems; multiobjective optimization problems will be dealt with in Chapter 4.2.

Preference Structures

- The function *f* together with the partially ordered set *(Z, rel)* defines a preference structure on the decision space *X* that reflects which solutions the decision maker / user prefers to other solutions.
- The preference structure $prefrel \subseteq X \times X$ is a binary relation with

  x_1 *prefrel* $x_2 :\Leftrightarrow f(x_1)$ *rel* $f(x_2)$

  The pair *(X, prefrel)* is an preordered set, but not necessarily a partially ordered set because different solutions may be mapped to the same objective vector and antisymmetry is not fulfilled (indifferent solutions).
- One says:
  - Two solutions x_1, x_2 are equal iff $x_1 = x_2$;
  - A solution x_1 is indifferent to a solution x_2 iff x_1 *prefrel* x_2 and x_2 *prefrel* x_1 and $x_1 \neq x_2$;
  - A solution x_1 is preferred to a solution x_2 iff x_1 *prefrel* x_2;
  - A solution x_1 is strictly preferred to a solution x_2 iff x_1 *prefrel* x_2 and not (x_2 *prefrel* x_1);
  - A solution x_1 is incomparable to a solution x_2 iff neither x_1 *prefrel* x_2 nor x_2 *prefrel* x_1.

Background: Binary Relations

- A binary relation *rel* over a set *S* is a subset of $S \times S$, i.e., *rel* contains ordered pairs of *S*. One usually writes *a rel b* instead of $(a, b) \in rel$.

- *rel* is called
  1. reflexive iff *a rel a* for all $a \in S$;
  2. irreflexive iff (not *a rel a)* for all $a \in S$;
  3. symmetric iff *a rel b* implies *b rel a* for all $a, b \in S$;
  4. asymmetric iff *a rel b* implies (not *b rel a)* for all $a, b \in S$;
  5. antisymmetric iff (*a rel b* and *b rel a)* implies $a = b$ for all $a, b \in S$;
  6. transitive iff (*a rel b* and *b rel c)* implies *a rel c* for all $a, b, c \in S$;
  7. connected iff *a rel b* or *b rel a* for all $a, b \in S$ with $a \neq b$;
  8. strongly connected iff *a rel b* or *b rel a* for all $a, b \in S$.

- A binary relation *rel* over a set *S* is an equivalence relation iff it is reflexive, symmetric, and transitive. An equivalence relation *rel* partitions the set *S* into disjoint subsets $S_1, ..., S_n$, the so-called equivalence classes, where for all a,b $\in S_i$ holds *a rel b*.

Background: Orders and Ordered Sets

- A binary relation *rel* over a set *S* is denoted as
  1. preorder iff *rel* is reflexive and transitive;
  2. partial order iff *rel* is a preorder and *rel* is antisymmetric;
  3. total preorder iff *rel* is a preorder and *rel* is connected;
  4. total order iff *rel* is a total preorder and *rel* is antisymmetric.

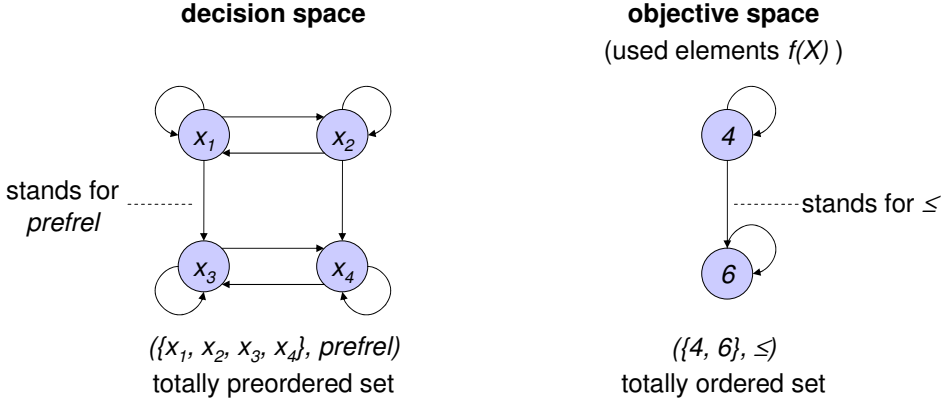
- The pair *(S, rel)* is called
  1. preordered set iff $rel \subseteq S \times S$ and *rel* is a preorder;
  2. totally preordered set iff $rel \subseteq S \times S$ and *rel* is a total preorder;
  3. partially ordered set iff $rel \subseteq S \times S$ and *rel* is a partial order;
  4. totally ordered set iff $rel \subseteq S \times S$ and *rel* is a total order.

- Given a partially ordered set *(S, rel)*, the element *a* of a subset *T* of *S* is called a minimal element of *T* iff *b rel a* implies $b = a$ for all $b \in T$.

  For a totally ordered set, the minimal element is unique, while for a partially ordered set there may exists several minimal elements.

The Relation Between Decision and Objective Space

Example: $(\{x_1, x_2, x_3, x_4\}, \aleph_0, f, \leq)$ with $f(x_1) = f(x_2) = 4$, $f(x_3) = f(x_4) = 6$

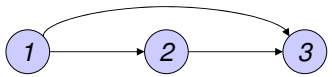


x_1 and x_2 as well as x_3 and x_4 are indifferent to each other.

Background: Graphs and Visualization of Preordered Sets

- A (directed) graph *G* is a tuple $G = (V, E)$ where
  - *V* stands for a set of vertices or nodes, and
  - $E \subseteq V \times V$ contains the edges that represent connections between pairs of vertices.

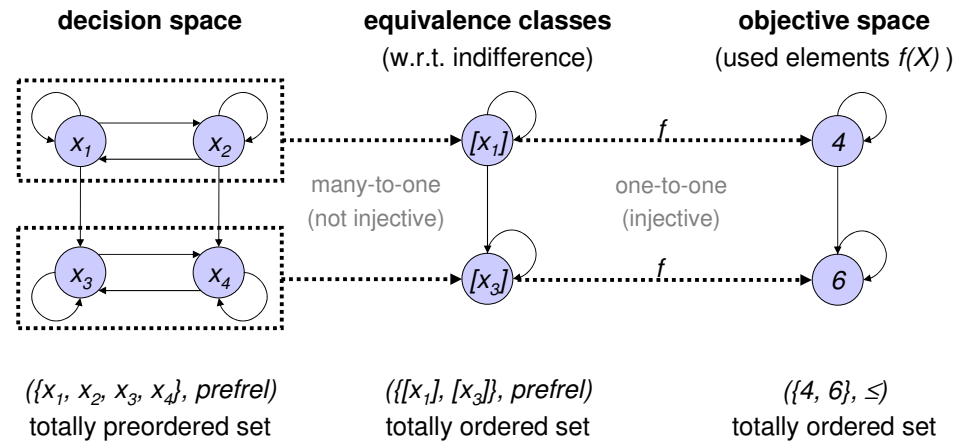
  Graphs are used as abstract representations for various things such as computer networks, metabolic pathways, etc. Vertices are usually drawn as circles and edges as arrows; for instance, the graph $G = (\{1,2,3\}, \{(1,2), (2,3), (1,3)\})$ can be depicted as



- Graphs can be naturally used to represent a preordered set *(S, rel)* where the elements of *S* are the nodes ($V = S$) and the edges stand for the pairs included in *rel* ($E = rel$). Often, a more compact representation is achieved by considering the cover relation $cov_{rel} := \{(a, b) \in rel \mid (b, a) \notin rel \wedge \forall c \in S : (a, c), (c, b) \in rel \Rightarrow (c, a) \in rel \vee (b, c) \in rel\}$. Example: graphs for $(\{1, 2, 3\}, \leq)$, left, and $(\{1, 2, 3\}, cov_{\leq})$, right:

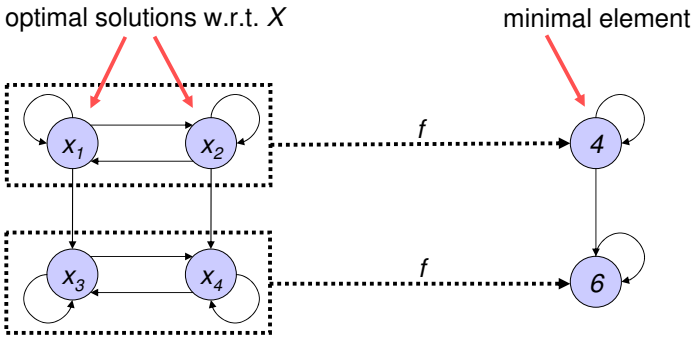
The Relation Between Decision and Objective Space

Example: $(\{x_1, x_2, x_3, x_4\}, \aleph_0, f, \leq)$ with $f(x_1) = f(x_2) = 4$, $f(x_3) = f(x_4) = 6$



The Notion of Optimality

- A solution $x \in X$ is called optimal with respect to a set $S \subseteq X$ iff no solution $x' \in S$ is strictly preferred to x, i.e., for all $x' \in S$: x' *prefrel* $x \Rightarrow x$ *prefrel* x'.
- In other words, $f(x)$ is a minimal element of $f(S)$ regarding the partially ordered set (Z, rel).



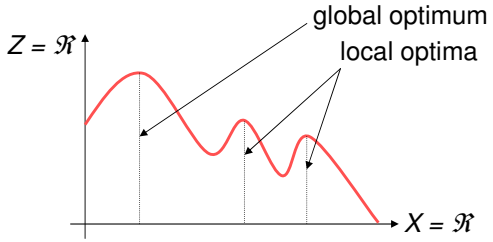
Global and Local Optima

One can distinguish two types of optimal solutions:

1. A global optimum is a solution $x \in X$ that is optimal w.r.t. X.
2. A local optimum is a solution $x \in X$ that is optimal within a certain neighboorhood of x given a distance metric d on X:

$x \in X$ is locally optimal $:\Leftrightarrow$
$\exists\, \varepsilon > 0$ with $S := \{x' \in X \mid d(x,x') \leq \varepsilon\}$ such that
(i) the neighborhood $S \setminus \{x\}$ is not empty
(ii) x is optimal w.r.t. the neighborhood S

Example:
$(\Re, \Re, f, \geq)$

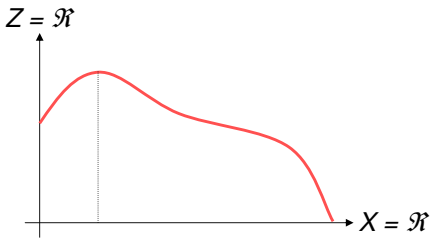


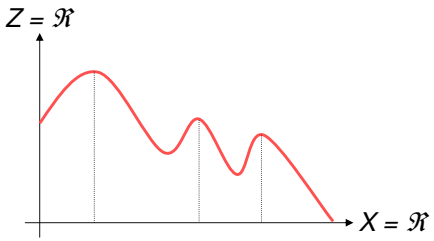
Unimodality and Multimodality

Sometimes problem landscapes are characterized according to whether there is only a single peak or several peaks exists; most real-world optimization problem, though, are multimodal.

Unimodal optimization problem:
All locally optimal solutions are indifferent to each other (they are at the same time globally optimal)

Multimodal optimization problem:
There exist at least two locally optimal solutions that are **not** indifferent to each other





The Goal of Optimization

Given a optimization problem (X, Z, f, rel),
the goal is to identify
a solution $x \in X$ that is globally optimal.

- Note that there may exist several global optima; we here assume that any global optimum is sufficient to solve the problem. If one is interested in finding all global optima, a different optimization problem emerges where the decision space consists of sets of solutions.
- With many practical applications, it is infeasible to generate a globally optimal solution; instead, one is often interested in finding an approximate solution that is as 'close' as possible to any global optimum.

Example: The Traveling Salesman Problem (TSP)

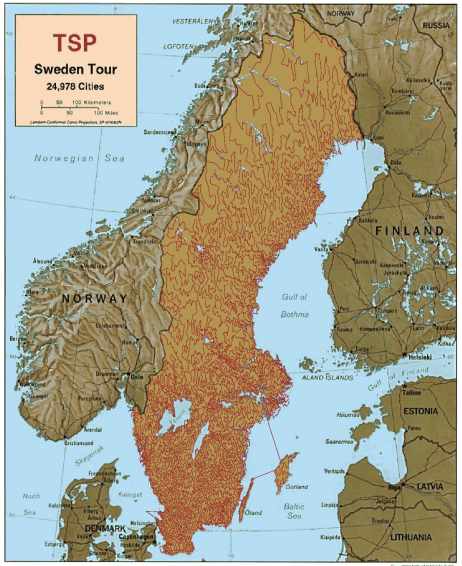
Given:

A number of cities and the traveling distances from any city to any other city

Question:

What is the shortest round-trip route that visits each city once and then returns to the starting city?

Note: many TSP variants exist



http://www.tsp.gatech.edu/

The Traveling Salesman Problem Formalized

Given: number *n* of cities, and traveling distances *d(i, j)* from city *i* to city *j*

Optimization problem: *(X, ℵ, f, ≤)* with

- $X = \{\pi \mid \pi : \{2,\ldots,n\} \rightarrow \{2,\ldots,n\} \text{ is a permutation over } \{2,\ldots,n\}\}$
- $f_{TSP}(\pi) = d(1,\pi(2)) + \left(\sum_{i=2}^{n-1} d(\pi(i),\pi(i+1))\right) + d(\pi(n),1)$

The influence of n on the size of the search space:

- In general: $|X| = (n-1)! \in O(n^n)$
- For a symmetric TSP where *d(i, j) = d(j, i)*, the number of different roundtrips is $(n-1)!/2$ as the direction does not matter
- The decision space grows rapidly with increasing n:
  $n = 10$: $|X| = 362880$
  $n = 20$: $|X| = 2.4329 \cdot 10^{18}$
  $n = 50$: $|X| = 3.0414 \cdot 10^{64}$
  (For comparison: the earth contains about 10^{21} liters of water)

What Makes Optimization Problems Difficult

The main factors (many more could be listed):

1. The size of the decision space is huge and prevents (advanced) enumeration techniques from being applicable,

and

2. either
   - the objective functions are known, but highly complex and/or poorly understood (NP hardness, non-linearities, non-differentiability, etc.)

   or
   - The objective functions are not given in closed form, but are determined numerically, by simulation, or even by experiment.

Background: Running Time and Order of Growth

- The running time of an algorithm on a given input is basically the number of elementary steps (arithmetic operations, branches, etc.) executed. In general, with each algorithm *A* there is a function r_A: $\aleph \rightarrow \aleph$ associated which gives the minimum (best-case running time), average (average-case running time), or the maximum (worst-case running time) number of elementary steps needed to process **any** input of size *n*.

- Often one is not interested in the exact formula for r_A, but rather in the asymptotic behavior or order of growth; here, one looks for the dominating terms in the formula for r_A that determine the growth of r_A, if *n* goes to infinity. For this reason, the *O*-notation has been introduced where *O(g(n))* defines a set of functions:

  $$O(g(n)) := \{h(n) \mid \exists c > 0, n_0 : 0 \leq h(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

  Roughly speaking, *O(g(n))* denotes the set of all functions that asymptotically grow not faster than *g(n)*, *g*: $\aleph \rightarrow \aleph$.

- One says the (worst-case) running time complexity of an algorithm *A* is of order *O(g(n))*, if $r_A \in O(g(n))$. For instance, an algorithm is said to have a linear or quadratic running time complexity, iff $r_A \in O(n)$ resp. $r_A \in O(n^2)$.

Background: NP Hardness

- In general, an optimization problem is considered to be efficiently solvable, if there exists an algorithm *A* the running time complexity of which is of order $O(n^k)$, where *k* is a specific constant; in other words, the running time of *A* is bound by a polynomial. Although for large *k*, e.g., *k=20*, such an algorithm can become unusable in practice.

  The situation is even worse for algorithms that require in the worst case exponentially many steps w.r.t. the input length *n*, e.g., $O(2^n)$. In this case, already for small *n*, the algorithm may not stop in reasonable time.

- The term NP hard characterizes a class of search problems for which no polyonmial algorithm is known. Although it is an open research problem to prove the non-existence of polynomial algorithms for these problems, most experts worldwide are convinced that this is actually case. That means whenever an optimization problem has been shown to be NP hard, then we cannot expect that in general an optimal solution can be determined within polynomial running time. For instance, the TSP is NP hard.

Types of Optimization Methods

- **Exact algorithms:**
  guarantee to find an optimal solution
- **Approximation schemes:**
  guarantee to find a solution that is not worse than an optimal solution by an arbitrary factor of ρ that can be set by the user; the running time is usually exponential in ρ
- **ρ - approximation algorithms:**
  guarantee to find a solution that is not worse than an optimal solution by a constant factor of ρ that cannot be changed by the user
- **Heuristic algorithms:**
  do not give any guarantees about the quality of the generated solution, but "usually" generate reasonably good solutions reasonably fast

A Heuristic Method for the TSP

Principle: always travel in direction of the closest city not visited yet...

1: $C = \{2, \ldots, n\}$
2: $a = 1$
3: $i = 2$
4: **while** $C \neq \emptyset$ **do**
5: choose $b \in C$ such that $d(a, b)$ is minimum
6: $C = C \setminus \{b\}$
7: $\pi(i) = b$
8: $a = b$
9: $i = i + 1$
10: **end while**
11: **return** π

Example:

d(i, j)	1	2	3	4
1	-	*2*	*4*	*5*
2	*2*	-	*3*	*7*
3	*4*	*3*	-	*23*
4	*5*	*7*	*23*	-

Heuristic algorithm:
route = 1-2-3-4-1
$f_{TSP} = 2 + 3 + 23 + 5 = 33$

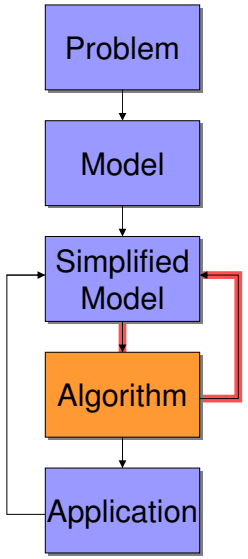
Exact algorithm:
route = 1-3-2-4-1
$f_{TSP} = 4 + 3 + 7 + 5 = 19$

Problem Solving Approaches

- The classical way of approaching an optimization problem is algorithm orientied: The problem is analyzed and then a problem-specific algorithm is designed.
  - It may be necessary to simplify the problem (model).
  - Requires both expertise and time.
- With the advent of sufficient computing resources, an alternative, model-oriented approach emerged: the model is refined until the relevant aspects of reality are appropriately taken into consideration and then a general search algorithm is applied.
  - The algorithm may perform poorly.
  - No guarantees about the quality of the generated solutions.

Often, it is helpful to use both approaches simultaneously.

The Algorithm-Oriented Approach



Focus: a good algorithm…

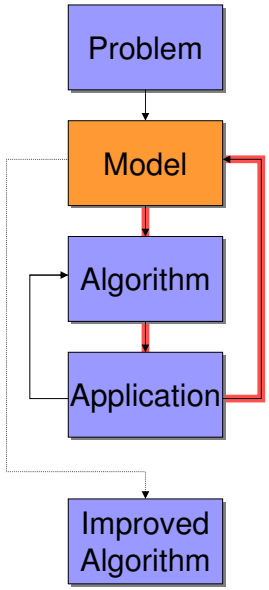
❶ Design of a problem-specific algorithm using known concepts:

- Dynamic programming
- Branch and bound
- Divide and conquer

❷ Usage of a general search algorithm with model constraints:

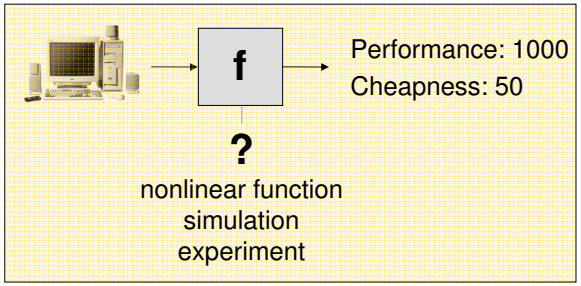
- Integer linear programming
- Gradient methods

The Model-Oriented Approach



Focus: a good model…

Usage of black-box optimization methods to approximate the optima:



- Evolutionary algorithms
- Randomized search algorithms

References

- T. Cormen, C. Leiserson, R. Rivest (1990): Introduction to Algorithms, MIT Press, Cambridge, MA. **(Chapters 1 + 2, running time and order of growth)**
- M. Ehrgott (2005): Multi-Criteria Optimization. Springer, Berlin. **(Chapter 1, order theory and optimization problems)**
- Z. Michalewicz, D. Fogel (2004): How to Solve It: Modern Heuristics. Springer, Berlin. **(Chapter 4, greedy and exhaustive search methods)**