Foundations of Computing

Darmstadt University of Technology
Dept. Computer Science

Winter Term 2005 / 2006

Copyright © 2004 by
Matthias Müller-Hannemann and Karsten Weihe
All rights reserved

http://www.algo.informatik.tu-darmstadt.de/

Section 1

Introduction

©2004 M. Müller-Hannemann
K. Weihe

Focus of "Optimization Algorithms"

- Not on efficiently solvable problems but on $\mathcal{NP}$–hard problems.
- Not on algorithms tailored to specific problems but on all–purpose algorithms (*generic* algorithms).
- These algorithms are not considered independently of each other but as specializations of even more general, fundamental algorithmic *concepts*.
- "Cool algorithms" such as evolution strategies and genetic algorithms will turn out to be (more or less straightforward) specializations.

©2004 M. Müller-Hannemann
K. Weihe

Overview of Section 1

Before going into the algorithms, we first focus for a while on the problems:

- examples, examples, examples, and
- some general aspects that all optimization problems have in common.

Remark:

The examples section overlaps with the annual lecture "Algorithmic Modeling" to some extent.

©2004 M. Müller-Hannemann
K. Weihe

Section 1.1:

Examples, Examples, Examples

Example 1: (Euclidean) TSP

- *Input*: a finite set of points in the plane.
- *Feasible output*: a closed cycle on the points.
- *Objective*: minimizing the length of the cycle (=sum of edge lengths).

General TSP:

- The problem from the last slide is but a special case of the TSP.

  $\longrightarrow$ Called the *Euclidean* TSP.

- The general TSP abstracts from points in the plane:
  - *Input*: a nonnegative integer n and a real–valued $(n \times n)$–matrix D.
  - *Feasible outputs*: the permutations of $\{1, \ldots, n\}$.
  - *Objective*: find a permutation σ that minimizes

$$\sum_{i=1}^{n-1} D[\sigma(i), \sigma(i+1)] + D[\sigma(n), \sigma(1)] \, .$$

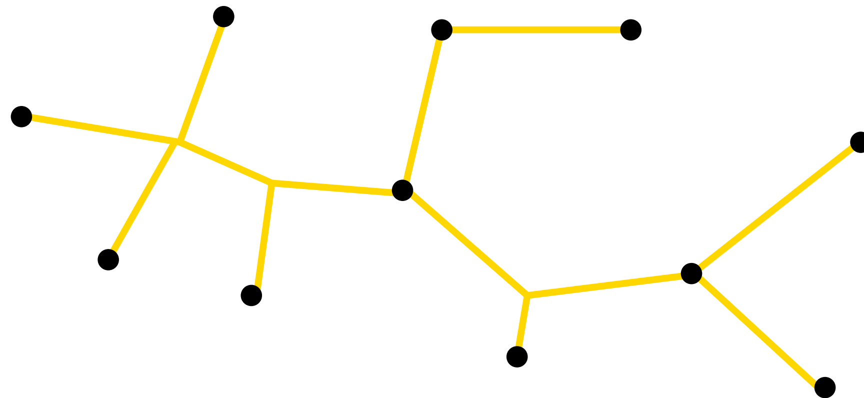
- *Euclidean TSP*: $D[i, j]$ is the (Euclidean) distance from point no. i to point no. j.

©2004 M. Müller-Hannemann K. Weihe

Example 2: Steiner tree

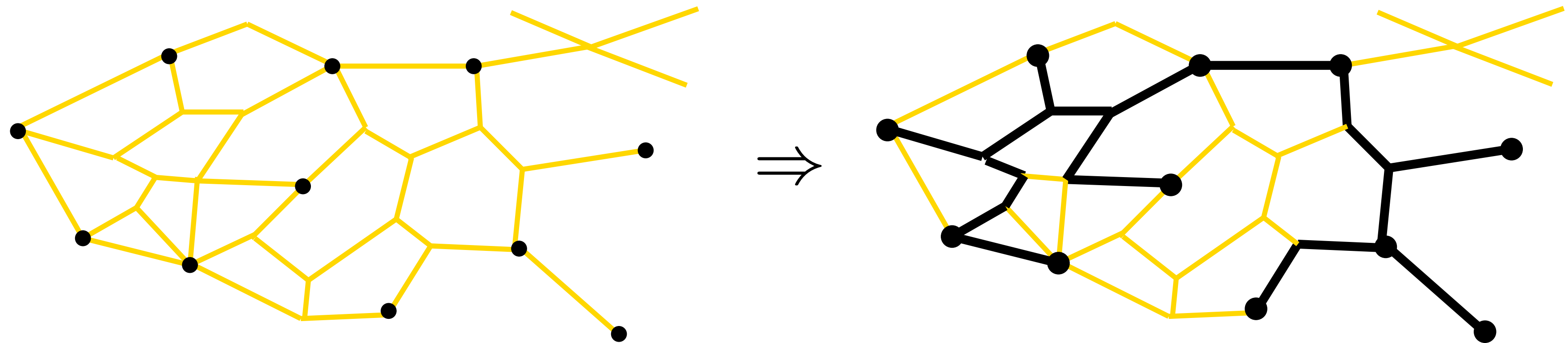
Steiner trees in the plane:

- *Input*: a finite set of points in the plane (*terminals*).
- *Desired output*: a "network" of shortest total length (=sum of edge lengths) such that all terminals are connected to each other in this network.



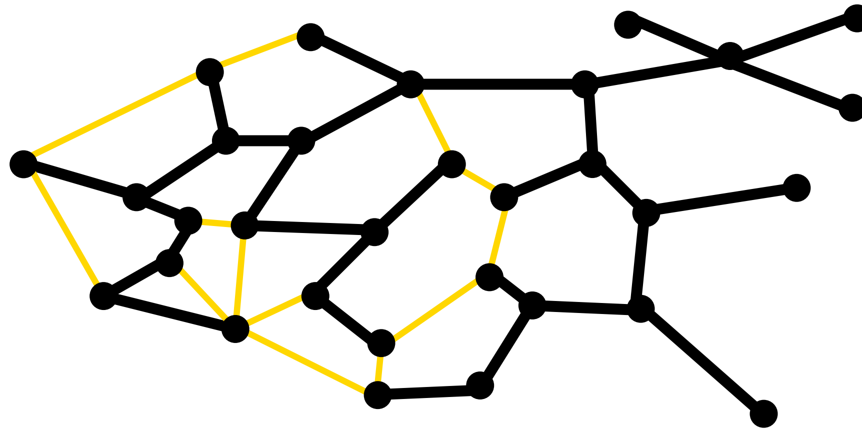
Steiner trees in networks

- *Input*: an undirected graph, a length for each edge, and a set of terminals (=selection of nodes of the graph).
- *Feasible output*: The output network must contain all terminals, be connected, and be part of this input network.
- *Objective*: minimizing the sum of the lengths of the used edges.



Fundamental special case: MST

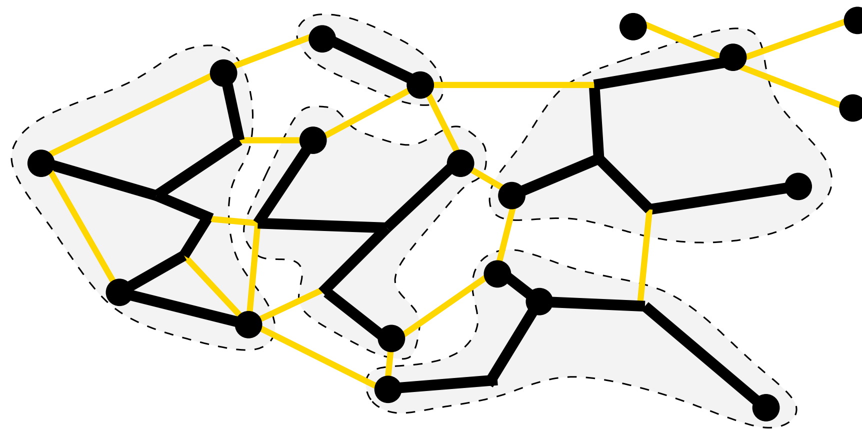
- *MST* = Minimum Spanning Tree.
- A proper special case of the Steiner–tree problem in networks.
- *Specialization*: *All* nodes of the input graph are terminals.
- Surprisingly,
  - the MST problem is polynomial,
  - the general Steiner–tree problem in networks is $\mathcal{NP}$–hard.



©2004 M. Müller-Hannemann K. Weihe

Generalization: Steiner–packing problem

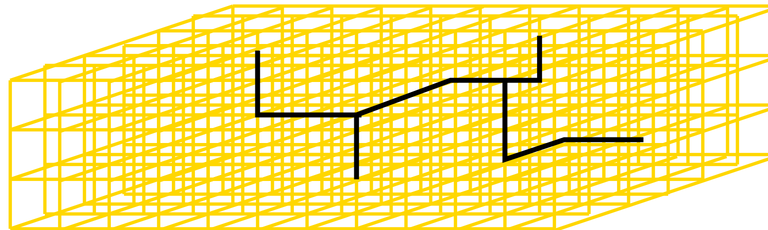
- Now we have several, pairwise disjoint sets of terminals.
- We are looking for one separate Steiner tree for each set of terminals.
- The Steiner trees must not hit each other.
- Sensible generalized objectives: sum or maximum of the total lengths of all individual Steiner trees.



©2004 M. Müller-Hannemann
K. Weihe

Application of this packing version: VLSI

- What's in a Name: VLSI = Very Large Scale Integrated.
- *Input*:
  - Graph: 3D–grid with a huge width and breadth but a small height.
  - Terminals: pins to be connected by wires.
- The connections are Steiner trees in the grid.
- Of course, these connections must not hit each other to avoid short–cuts.



©2004 M. Müller-Hannemann K. Weihe

But...

- The total length of the tree is not the only relevant objective.
- The pins are not completely exchangeable: One designated pin serves as a driver ("master"), which sends a signal to all other pins ("slaves") in its set.
- Important objective is (roughly) to minimize the maximal run time from each driver to all of its slaves.
- This objective is much harder.
- For certain stages of the VLSI design process, the objectives from Slide no. 10 are sensible alternatives:
  - much easier to handle mathematically,
  - probably (hopefully!) close enough to reality.

Variation: Fault–tolerant networks

- A communication network should not become disconnected if one server (node) or one connection (arc) breaks down due to hardware/software failures.
- In terms of graph theory: Any two nodes A and B should be connected through the network by at least two paths which are
  - edge–disjoint in case only break–downs of edges are relevant;
  - node–disjoint except for A and B (a.k.a. *internally* node–disjoint) in case break–downs of servers shall also be taken into account.
- Further realistic variants:
  - At least three, four... connecting, disjoint paths for any two nodes A and B.
  - Different numbers of required disjoint paths for pairs A, B with different priorities (e.g. important companies vs. people in the outback).

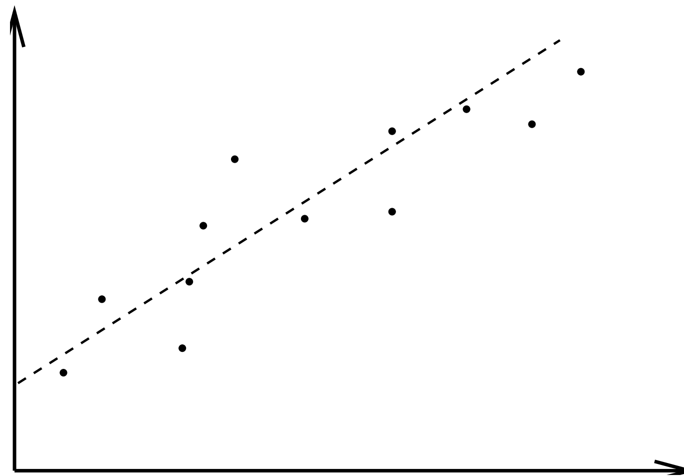
©2004 M. Müller-Hannemann K. Weihe

Example 3: timetabling

- *Given*:
  - A set of rooms, each coming with the number of seats.
  - A set of time slots, supposed to be non-overlapping to make things easier.
  - A set of courses, each filling exactly one time slot, to make things easier.
  - A set of students, each coming with a list of courses in which this student is registered.
- *Find*: an assignment of rooms and time slots to courses such that
  - no two courses are assigned the same room and time slot,
  - the audience of a course is not larger than the capacity of the room assigned to this course, and
  - no student is registered for two courses at the same time.

Example 4: linear regression

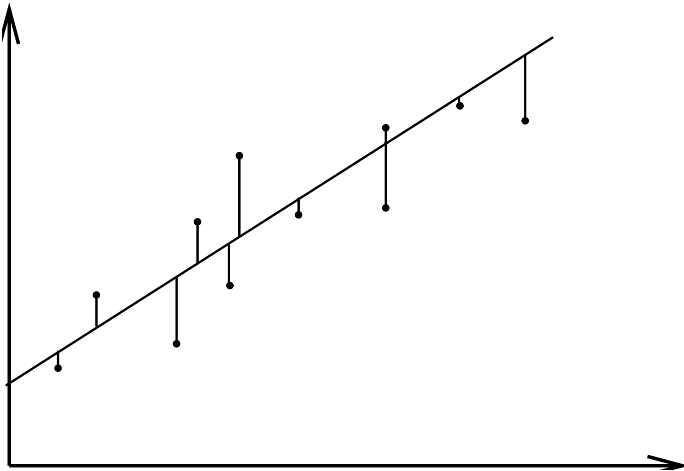
- *Input*: a finite set of points in the plane.
- *Background*: These points are probes which would ideally (i.e. without inaccuracies in the measured values) reveal an affine–linear relation between the two parameters (=lie on a straight line).
- *Desired output*: the "best approximating straight line".



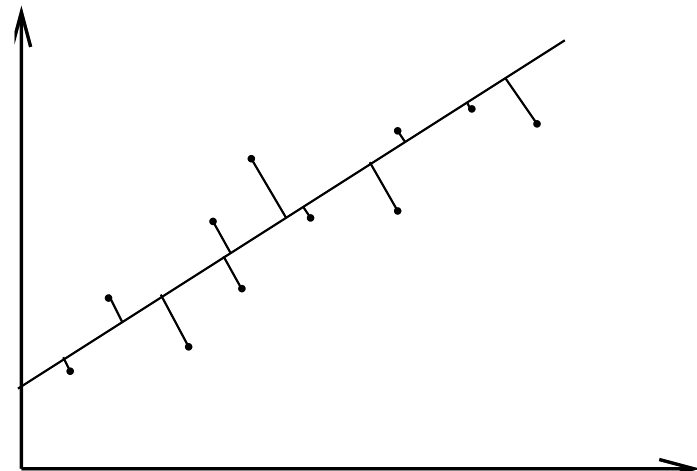
What is the “best approximating straight line”?

The most obvious, straightforward definition for “best approximating line” would be to minimize the sum of the distances of all points from the line.

Either in vertical direction:



Or in perpendicular direction:



©2004 M. Müller-Hannemann K. Weihe

Best approximating straight line (cont'd)

- The standard approach is known as the *least–squares method*.
- *This means*: The sum of the *squares* of the vertical distances is to be minimized.
- *Why this model*: Can be reduced to solving one linear equation system.

  $\longrightarrow$ Proof left out here (see lectures on statistics).

©2004 M. Müller-Hannemann
K. Weihe

Example 5: nonlinear optimization problem

Let f, g_i $(i = 1, \ldots, m)$, h_j $(j = 1, \ldots, p)$ be differentiable (or at least continuous) functions from $\mathbb{R}^n \mapsto \mathbb{R}$. Then the problem

$$\begin{gathered} \text{minimize } f(x) \text{ subject to} \\ g_i(x) \leq 0 \quad \text{for } i = 1, \ldots, m \\ h_j(x) = 0 \quad \text{for } j = 1, \ldots, p \\ x \in \mathbb{R}^n \end{gathered}$$

is called a *nonlinear optimization problem*.

©2004 M. Müller-Hannemann K. Weihe

Example 6: convex optimization problem

- Given two distinct points $x, y \in \mathbb{R}^n$, a *convex combination* of them is any point of the form $z = \lambda x + (1 - \lambda) y$ for $\lambda \in \mathbb{R}$ and $0 \leq \lambda \leq 1$ (this is a *strict convex combination* if $0 < \lambda < 1$).
- A set $S \subseteq \mathbb{R}^n$ is *convex* if it contains all convex combinations of pairs of points $x, y \in S$.
- A function $f : S \mapsto \mathbb{R}$ (where $S \subseteq \mathbb{R}^n$ is a convex set) is *convex* in S if for any two points $x, y \in S$ and $0 \leq \lambda \leq 1$ we have

$$\lambda f(x) + (1 - \lambda) f(y) \geq f(\lambda x + (1 - \lambda) y).$$

- Examples of convex functions: linear, quadratic, exponential.
- Let $S \subseteq \mathbb{R}^n$ be a convex set, and $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a convex function. Then the problem of finding an $x \in S$ that minimizes $f(x)$ among all $x \in S$ is called a *convex minimization problem*.

©2004 M. Müller-Hannemann K. Weihe

Example 7: linear programming problem

Given $c \in \mathbb{R}^n, A \in \mathbb{R}^{(m,n)}, b \in \mathbb{R}^m$, then

$$\begin{gathered}\text{maximize } c^T x \text{ subject to}\\ Ax \le b\\ x \in \mathbb{R}^n\end{gathered}$$

is called a *linear programming problem* (LP).

$\longrightarrow$ A special case of convex optimization.

©2004 M. Müller-Hannemann
K. Weihe

Example 8: integer linear programs

- Given $c \in \mathbb{R}^n, A \in \mathbb{R}^{(m,n)}, b \in \mathbb{R}^m$, then

$$\text{maximize } c^T x \text{ subject to}$$
$$Ax \leq b$$
$$x \in \mathbb{Z}^n$$

  is called an *integer linear programming problem* (ILP).

- If a variable is restricted to be zero-one valued, it is called a *binary variable*.

- If all variables are binary variables, the problem is called a *binary linear programming problem*.

- If only a subset of the variables is required to be integer-valued, the corresponding problem is called a *mixed integer linear programming problem (MILP)*.

©2004 M. Müller-Hannemann
K. Weihe

Example 9: semi–definite programming

- This is a specific class of algorithmic problems or, better, of *models* or *formulations* of algorithmic problems.
- In semi–finite models,
  - the number of variables is finite but
  - the number of side constraints may be infinite.

Exemplary problem class: min-max problems

- *Input*: two arbitrary sets, S and T, and an objective function $C : S \times T \longrightarrow \mathbb{R}$.
- *Feasible outputs*: pairs (s, t) such that $s \in S$ and $t \in T$.
- *Objective*: find $s_0 \in S$ and $t_0 \in T$ such that

$$C(s_0, t_0) = \min_{s \in S} \max_{t \in T} C(s, t) \,.$$

©2004 M. Müller-Hannemann K. Weihe

Semi–definite re–formulation of the example:

- Drop S and introduce a new variable $z \in \mathbb{R}$.

  $\longrightarrow$ The solution space is now $T \times \mathbb{R}$.

- For each $s \in S$ introduce the function

$$g_s : T \longrightarrow \mathbb{R}$$

  defined by $g_s(t) := C(s,t)$ for all $t \in T$.

- For each $s \in S$ introduce the side constraint

$$g_s(t) \leq z\,.$$

- The objective to be minimized is z.

$\longrightarrow$ If T is finite–dimensional, this re–formulation is semi–definite.

©2004 M. Müller-Hannemann K. Weihe

Concrete application: polynom approximation

- *Input*: a real–valued interval $[a \dots b]$, a function $f : [a \dots b] \longrightarrow \mathbb{R}$, a natural number $k \in \mathbb{N}_0$.
- *Feasible output*: a real–valued vector $u = (u[0], u[1], \dots, u[k]) \in \mathbb{R}^{k+1}$.
- *Auxiliary notation*: for $u \in \mathbb{R}^{k+1}$ let

$$\Delta(u) := \max_{x \in [a \dots b]} \left| f(x) - \sum_{i=0}^{k} u[i] \cdot x^i \right| .$$

- *Objective*: find $u_0 \in \mathbb{R}^{k+1}$ such that

$$\Delta(u_0) = \min_{u \in \mathbb{R}^{k+1}} \Delta(u) .$$

Informal interpretation:

- The function f is to be approximated by a polynom of degree at most k such that the maximal approximation error in $[a \dots b]$ is minimized.
- It is well known that a polynom of degree at most k can be identified with its $k+1$ coefficients, so we may optimize over $\mathbb{R}^{k+1}$.

©2004 M. Müller-Hannemann K. Weihe

Semi–finite re–formulation of polynom approximation:

- *Input*: as before.
- *Feasible output*: a pair (u, z) such that $u \in \mathbb{R}^{k+1}_{,}$ $z \in \mathbb{R}$, and

$$\left| f(x) - \sum_{i=0}^{k} u[i] \cdot x^i \right| \leq z \text{ for all } x \in [a \ldots b] .$$

- *Objective*: minimizing z.

Section 1.2:

General Discussion of Algorithmic Problems

©2004 M. Müller-Hannemann
K. Weihe

Types of algorithmic problems

- Decision problem: is there a solution or not.
- Construction problem: if there is a solution, construct one.
- Enumeration problem: give all solutions.
- Optimization problem: construct a solution that is optimal (or at least nearly optimal) subject to a given objective.

Construction and optimization problems

- Construction and optimization problems are by far the most important types of problems in practice.
- Algorithms for decision problems are typically *constructive*, which means they solve the corresponding construction problem as well.
- Trivially, each construction problem may be viewed as an optimization problem: just define an objective that assigns the same alue to each solution.
- Moreover, many generic algorithms are in fact applicable to optimization problems only.
- To apply such a generic algorithm to a construction problem, it has to be transformed into an optimization problem (later on, we will see general techniques for that).
- For all of these reasons, we may focus on optimization problems in the following.

Ingredients of an Optimization Problem

An optimization problem may be described by the following ingredients:

- the feasible inputs (a.k.a. feasible *instances*),
- the feasible outputs for a given instance,
- the objective.

©2004 M. Müller-Hannemann K. Weihe

More formally:

- A set $\mathcal{I}$ of potential inputs.
- For each input $I \in \mathcal{I}$ a set F_I of *feasible solutions*.
- An objective function $C_I : F_I \longrightarrow \mathbb{R}$ and a direction: minimizing or maximizing.
- task:
  - determine whether $F_I \neq \emptyset$ and,
  - if so, find $x \in F_I$ such that

$$C_I(x) = \begin{cases} \min\{C_I(y) \mid y \in F_I\} \\ \max\{C_I(y) \mid y \in F_I\} \end{cases}$$

©2004 M. Müller-Hannemann K. Weihe

Example 10: matching

Matching:

- *Input*: an undirected graph $G = (V, E)$.
- *Feasible output*: a set $M \subseteq E$ such that no two edges in M have an incident node in common.

  $\longleftrightarrow$ M is called a *matching* of G.
- *Objective*: maximizing $\#M$.

Ingredients:

- The instances $I \in \mathcal{I}$ are the undirected graphs $G = (V, E)$.
- For an instance $G = (V, E) \in \mathcal{I}$, F_G is the set of all matchings in G.
- The objective function counts the number of edges in the matching.

©2004 M. Müller-Hannemann K. Weihe

Specification of feasible solutions:

- Typically, the sets F_I are specified by ground sets S_I and side constraints.
- *Side constraint*: a predicate (=boolean function) on S_I.
- For an instance $I \in \mathcal{I}$, let $\mathcal{SC}_I$ denote the set of all side constraints for I.
- *Feasible solution*: the set of $x \in S_I$ such that $c(x)$ is satisfied for all $c \in \mathcal{SC}_I$.

Applied to the matching example:

- For $G = (V, E) \in \mathcal{I}$, we can define S_G as the power set (=set of all subsets) of E.
- *Side constraints*: For each subset $M \subseteq E$ and for all $\{v_1, v_2\}, \{w_1, w_2\} \in M$, it must be

$$v_1 \neq w_1,\ v_1 \neq w_2,\ v_2 \neq w_1, \text{ and } v_2 \neq w_2.$$

©2004 M. Müller-Hannemann K. Weihe

More formal specification of the matching problem:

- For an instance $G = (V, E)$ of the matching problem, the elements of S_G may be alternately encoded as the set of all 0/1–vectors x defined on the index set E.

- *Side constraints*: For $x \in S_G$ and all $v \in V$, it must be

$$\sum \left\{ x[\{w_1, w_2\}] \,\middle|\, w_1,\, w_2 \in V;\, \{w_1, w_2\} \in M;\, w_1 = v \vee w_2 = v \right\} \;\leq\; 1\,.$$

©2004 M. Müller-Hannemann K. Weihe

More complex example: the general TSP revisited

- $\mathcal{I}$ can be viewed as the set of all quadratic real–valued matrices D:

  $D[i, j] =$ distance from point no. i to point no. j.

  $\longrightarrow$ Cf. Slide no. 6.

- For an $(n \times n)$–matrix $I \in \mathcal{I}$, S_I may then be the set of all quadratic 0/1–matrices X of size n:

  $X[i, j] = 1 \iff$

  j follows i immediately on the cycle corresponding to X.

- *Objective* for matrix size n:

$$\text{minimizing} \sum_{i=1}^{n} \sum_{j=1}^{n} D[i, j] \cdot X[i, j].$$

©2004 M. Müller-Hannemann
K. Weihe

Exemplary set of side constraints:

- $\forall i \in \{1, \ldots, n\} : X[i,i] = 0$

  $\longrightarrow$ No arc from a node to itself.

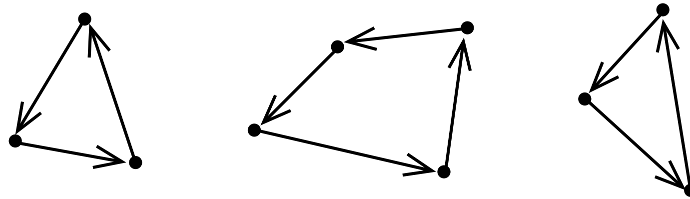
- $\forall i \in \{1, \ldots, n\} : \sum_{j=1}^{n} X[i,j] = 1$

  $\longrightarrow$ Each node is left by exactly one arc.

- $\forall i \in \{1, \ldots, n\} : \sum_{j=1}^{n} X[j,i] = 1$

  $\longrightarrow$ Each node is approached by exactly one arc.

However, still possible:



Additional side constraints to fix the problem:

$$\forall S \subsetneq \{1, \ldots, n\},\, S \neq \emptyset : \sum_{\substack{i=1 \\ i \in S}}^{n} \sum_{\substack{j=1 \\ j \notin S}}^{n} X[i,j] \geq 1$$

$\longrightarrow$ At least one arc from each (non–empty) set S of nodes to its (non–empty) complement.

See the annual lecture on “algorithmic modeling” (every winter term) for more stuff like this.

Feasibility and Boundedness

- An instance $I \in \mathcal{I}$ of an algorithmic problem is called *feasible*, if $F_I \neq \emptyset$, otherwise *infeasible*.
- An instance $I \in \mathcal{I}$ of a minimization (resp. maximization) problem is called *bounded*, if the objective function is bounded over F_I from below (resp. above), otherwise, it it called *unbounded*.

Note:

- Boundedness of an instance $I \in \mathcal{I}$ is not identical with the normal, set–theoretic boundedness of F_I.
- Consider the important case that $F_I \subseteq \mathbb{R}^n$ for some n, F_I is closed, and the objective function is continuous on F_I:
  - Obviously, boundedness of F_I implies boundedness of I in this case.
  - Basic calculus says for this case: boundedness of $f(I)$ implies the existence of a minimum and a maximum.

    $\longrightarrow$ In particular, boundedness of F_I.

©2004 M. Müller-Hannemann K. Weihe

Complexity

- *Methodological problem*: Most optimization problems (as well as any other kind of algorithmic problems) from practice are $\mathcal{NP}$–hard.
- From lectures on “theoretical computer science”, we (should!?) know what this means in *theory*.
- However, what does it mean in *practice*?

Answer:

If an optimization problem is $\mathcal{NP}$–hard, there is (most probably) no algorithm in the world for which the following two conditions can be guranteed simultaneously:

- It computes an optimal solution (=does its job fully accurately).
- For a reasonable input size, it does not require an astrononomic amount of run time.

©2004 M. Müller-Hannemann K. Weihe

Note:

in this context, an *astronomic* amount of time typically means that the (estimated) time limits of this universe are exceeded by orders of magnitude even for worst–case instances of rather small sizes.

Enforcing a reasonable amount of time:

- Certain algorithms compute an optimal solution by computing a large sequence of feasible solutions.

  $\longrightarrow$ First of all (yet not exclusively), local search algorithms (will be discussed in Section 2).

- Such an algorithm can be terminated after a fixed amount of time.

  $\longrightarrow$ The output is the best feasible solution computed so far.

- Clearly, this is at the cost of a worse objective value of the output.

©2004 M. Müller-Hannemann K. Weihe

Finding feasible solutions:

- For some optimization problems, it is easy to find a feasible solution.
  $\longrightarrow$ TSP, Steiner tree, ...
- For other optimization problems, even this task is $\mathcal{NP}$–hard.
  $\longrightarrow$ Timetabling (proof left out).

Consequence:

If an optimization problem is of the second kind, there is (most probably) no algorithm in the world for which the following two conditions can be guranteed simultaneously:

- It computes a *feasible* (sic!) solution.
- Depending on the input, it does not require an astrononomic amount of run time.

Exact vs. Approximation vs. Heuristic

- An algorithm is called *exact* if:
  - *Feasibility version*: It *provably* finds a feasible solution if there is one.
  - *Optimization version*: It *provably* finds an optimal solution if there is one.
- An algorithm is called *approximative* if:
  - *Feasibility version*: It finds a solution that is *provably* not too far from feasibility according to some reasonable measure.
  - *Optimization version*: It finds a solution that is *provably* not too far from optimality according to some reasonable measure.
- An algorithm is called *heuristic* if:
  - *Feasibility version*: It attempts at finding a feasible or nearly feasible solution, but no quality guarantee is proved.
  - *Optimization version*: It attempts at finding an optimal or nearly optimal solution, but no quality guarantee is proved.

©2004 M. Müller-Hannemann K. Weihe