Introduction to Optimization

October 7

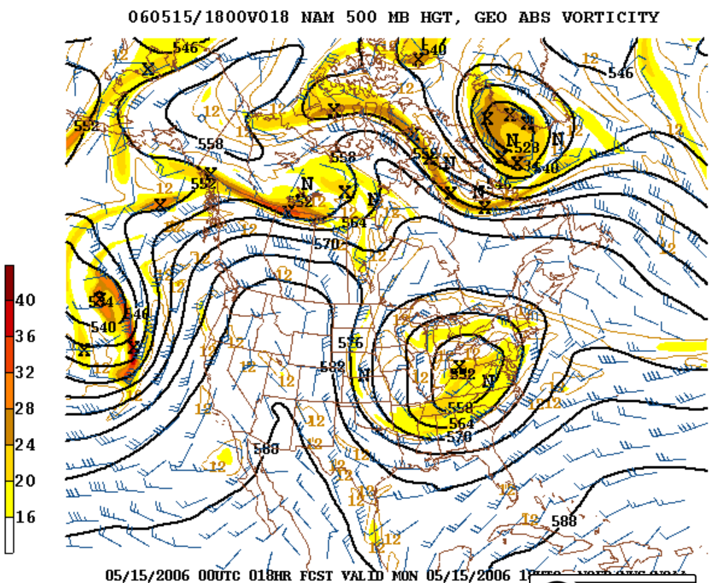
~~September 26~~, 2016

École Centrale Paris, Châtenay-Malabry, France

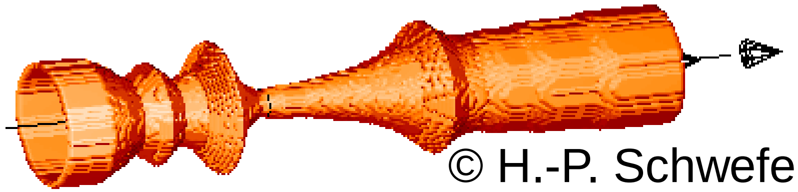
Dimo Brockhoff

Inria Saclay – Ile-de-France

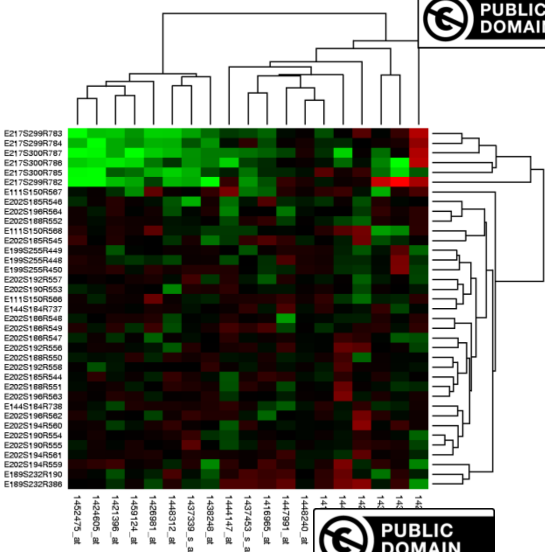
What is Optimization?



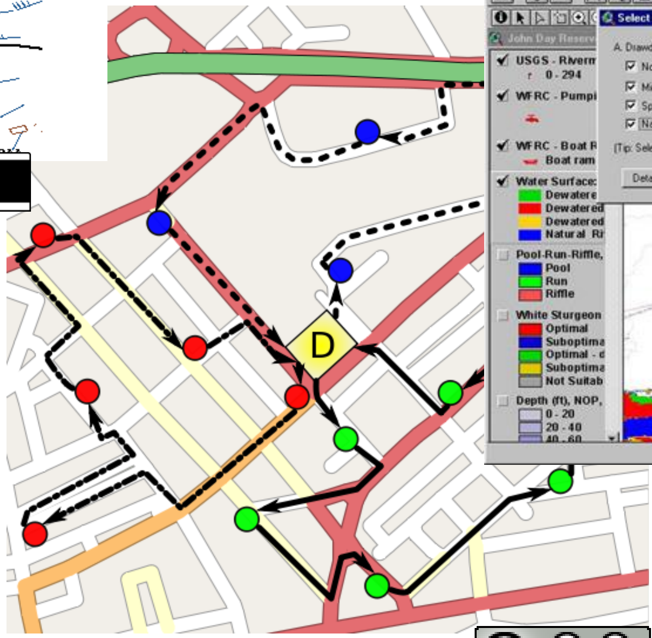
PUBLIC DOMAIN



© H.-P. Schwefel

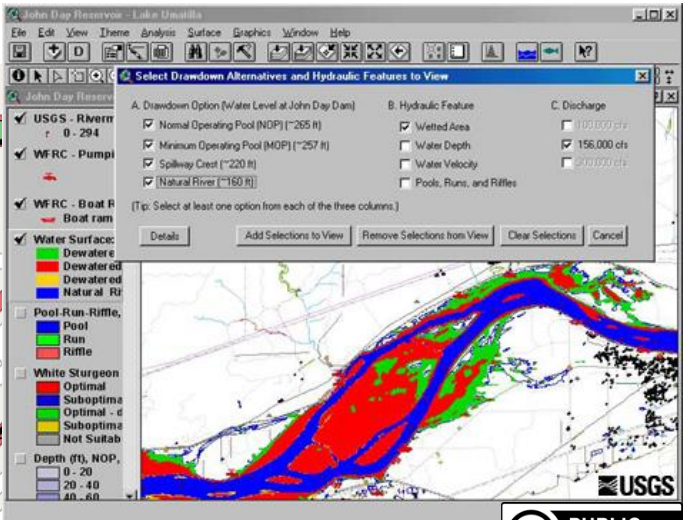


PUBLIC DOMAIN



Maly LOLek

CC BY SA



PUBLIC DOMAIN

What is Optimization?

- find solutions x which minimize f in the shortest time possible (maximization is reformulated as minimization) or
- find solutions x with as small f(x) as possible in the shortest time possible

Optimization problem: find the best solution among all feasible ones!

  - "minimize the function f!"

Search problem: output a solution with a given structure!

  - "find a tour through a given set of cities shorter than X km!"

Decision problem: is there a solution with a certain property?

  - "is n prime?"
  - "is there a clique in the graph of size at least 5?"

Example: Sorting

- Aim: Sort a set of cards/words/data
- Re-formulation: minimize the “unsortedness”

- E F C A D B
- B A C F D E
- A B C D E F

sortedness increases

Example: Sorting

Classical Questions:

- What is the underlying algorithm?
  (How do I solve a problem?)
- How long does it take to optimize?
  (How long does it take? Which guarantees can I give?)
- Is there a better algorithm or did I find the optimal one?

Course Overview

Date		Topic
Fri, 7.10.2016		Introduction
Fri, 28.10.2016	D	Greedy algorithms
Fri, 4.11.2016	D	Branch and bound
Fri, 18.11.2016	D	Dynamic programming
Mon, 21.11.2016 in S103-S105	D	Approximation algorithms and heuristics
Fri, 25.11.2016 in S103-S105	C	Introduction to Continuous Optimization I
Mon, 28.11.2016	C	Introduction to Continuous Optimization II
Mon, 5.12.2016	C	Gradient-based Algorithms
Fri, 9.12.2016	C	Stochastic Optimization and Derivative Free Optimization I
Mon, 12.12.2016	C	Stochastic Optimization and Derivative Free Optimization II
Fri, 16.12.2016	C	Benchmarking Optimizers with the COCO platform
Wed, 4.1.2017		Exam

all classes last 3h15 and take place in S115-S117 (see exceptions)

Remarks

- possibly not clear yet what the lecture is about in detail
- but there will be always examples and exercises to learn "on-the-fly" the concepts and fundamentals

Overall goals:

❶ give a broad overview of where and how optimization is used

❷ understand the fundamental concepts of optimization algorithms

❸ be able to apply common optimization algorithms on real-life (engineering) problems

Please ask questions
if things are unclear throughout the course!

The Exam

- Wednesday, 4th January 2017 from 09h45 till 12h45
- open book: take as much material as you want
- (most likely) combination of
  - questions on paper (to be handed in)
  - practical exercises (send source code and results by e-mail)

All information also available at

`http://researchers.lille.inria.fr/~brockhof/introoptimization/`

(exercise sheets, lecture slides, additional information, links, ...)

Remarks on Exercises

- included within the lecture (typically 1/3 of it)
- expected to be done in python
- hence, please make sure you have python installed on your laptop until the second lecture
- Anaconda is the recommended way to get there:

  https://www.continuum.io/downloads
- (basic) example solutions will be made available afterwards
- not graded but please see it as training for the exam

Overview of Today's Lecture

- More examples of optimization problems
  - introduce some basic concepts of optimization problems such as domain, constraint, ...
- Basic notations such as the O-notation
- Beginning of discrete optimization part
  - a brief introduction to graphs
  - concrete examples of problems used later on in the lecture

General Context Optimization

Given:

set of possible solutions — *Search space*

quality criterion — *Objective function*

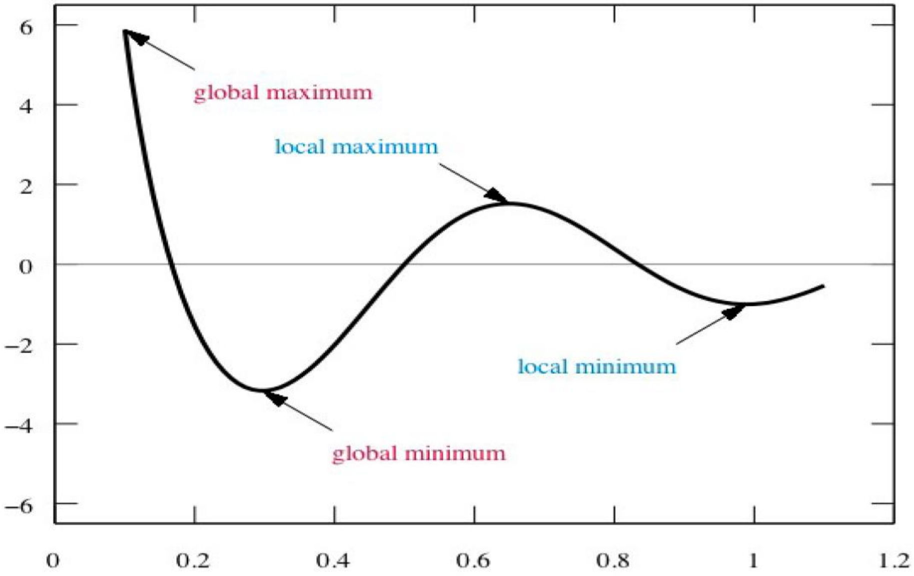
Objective:

Find the best possible solution for the given criterion

Formally:

Maximize or minimize

$$\mathcal{F} : \Omega \mapsto \mathbb{R},$$
$$x \mapsto \mathcal{F}(x)$$



Constraints

Maximize or minimize

$$\mathcal{F} : \Omega \mapsto \mathbb{R},$$
$$x \mapsto \mathcal{F}(x)$$

unconstrained

Ω

Maximize or minimize

$$\mathcal{F} : \Omega \mapsto \mathbb{R},$$
$$x \mapsto \mathcal{F}(x)$$
$$\text{where} \quad g_i(x) \leq 0$$
$$h_j(x) = 0$$

example of a constrained Ω

Constraints explicitely or implicitely define the feasible solution set

[e.g. ||x|| - 7 ≤ 0 vs. every solution should have at least 5 zero entries]

Hard constraints *must* be satisfied while soft constraints are preferred to hold but are not required to be satisfied

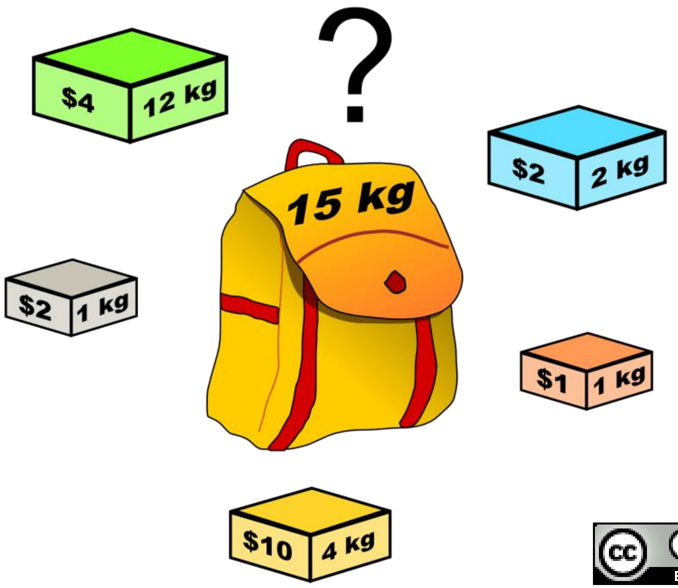
[e.g. constraints related to manufactoring precisions vs. cost constraints]

Example 1: Combinatorial Optimization

Knapsack Problem

- Given a set of objects with a given weight and value (profit)
- Find a subset of objects whose overall mass is below a certain limit and maximizing the total value of the objects

[Problem of ressource allocation with financial constraints]



Dake

Exercise: how would you formalize this problem?

❶ what is the search space?

❷ how do you write down the objective function?

❸ what are the constraints?

Knapsack Problem

- Given a set of objects with a given weight and value (profit)
- Find a subset of objects whose overall mass is below a certain limit and maximizing the total value of the objects

[Problem of ressource allocation with financial constraints]

Dake

$$\text{max.} \sum_{j=1}^{n} p_j x_j \text{ with } x_j \in \{0,1\}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq W$$

$$\Omega = \{0,1\}^n$$

Traveling Salesperson Problem (TSP)

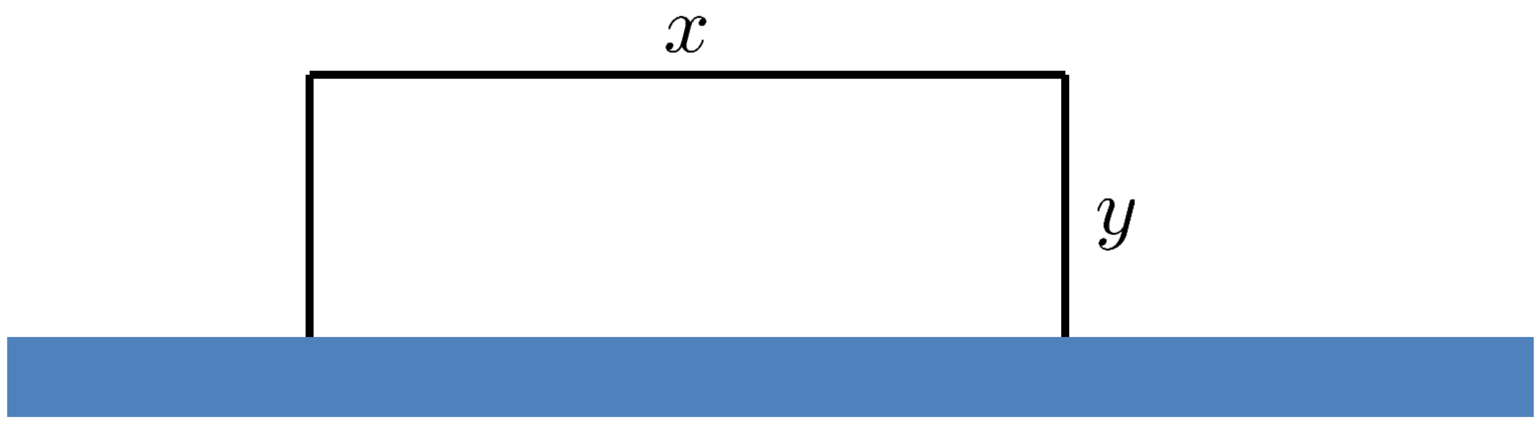
- Given a set of cities and their distances
- Find the shortest path going through all cities

PUBLIC DOMAIN

$$\Omega = S_n \text{ (set of all permutations)}$$

Example 3: Continuous Optimization

A farmer has 500m of fence to fence off a rectangular field that is adjacent to a river. What is the maximal area he can fence off?



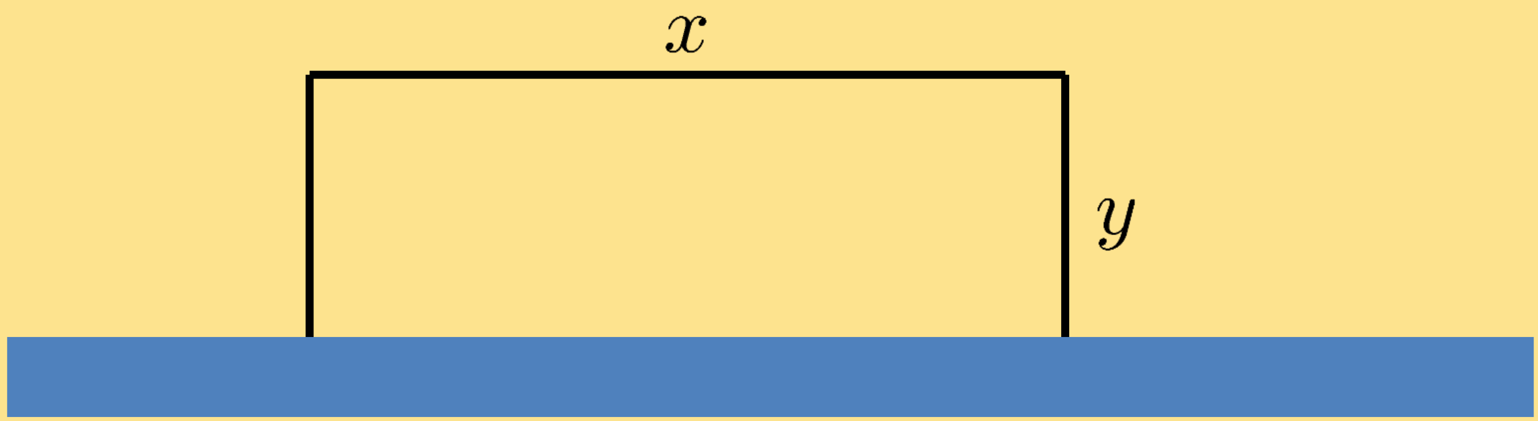
Exercise:

❶ how would you formalize this problem?

❷ how do you solve it? (it can be done analytically!)

Example 3: Continuous Optimization

A farmer has 500m of fence to fence off a rectangular field that is adjacent to a river. What is the maximal area he can fence off?



❶
$$\Omega = \mathbb{R}^2_{>0} :$$
$$\max xy$$
$$\text{where } x + 2y = 500$$

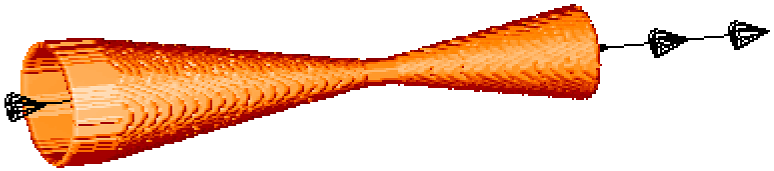
❷
$$\text{with } x = 500 - 2y:$$
$$\max f(x) = -2y^2 + 500y$$
$$\frac{d}{dx} f(x) = -4y + 500$$
$$\frac{d}{dx} f(x) = 0 \Leftrightarrow \begin{cases} y = 125 \\ (x = 250) \end{cases}$$

Example 4: A "Manual" Engineering Problem

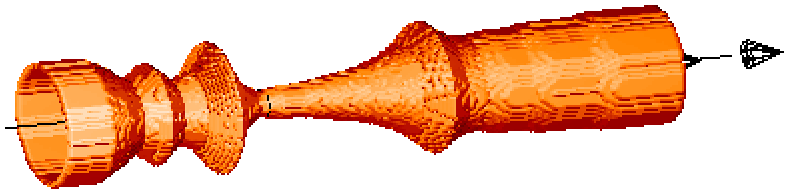
Optimizing a Two-Phase Nozzle [Schwefel 1968+]

- maximize thrust under constant starting conditions
- one of the first examples of Evolution Strategies

initial design:



final design:



$\Omega =$ all possible nozzles of given number of slices

copyright Hans-Paul Schwefel

[http://ls11-www.cs.uni-dortmund.de/people/schwefel/EADemos/]

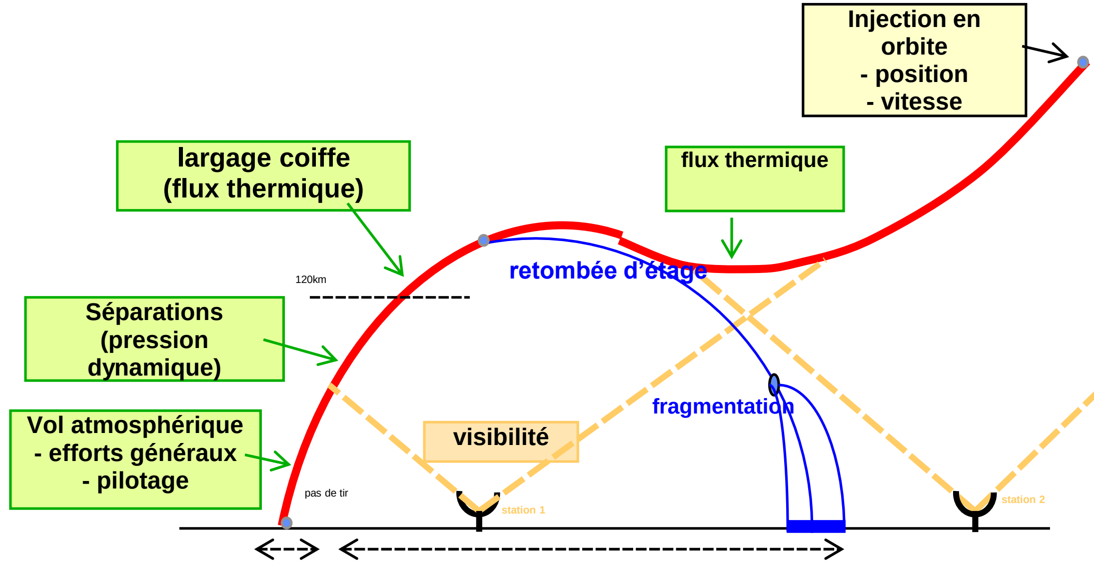
Example 5: Constrained Continuous Optimization

Design of a Launcher

BY SA

Poppy

$$\Omega = \mathbb{R}^{23}$$



copyright by Astrium

- Scenario: multi-stage launcher brings a satellite into orbit
- Minimize the overall cost of a launch
- Parameters: propellant mass of each stage / diameter of each stage / flux of each engine / parameters of the command law

23 continuous parameters to optimize

+ constraints

Example 6: History Matching/Parameter Calibration

One wide class of problems:

- matching existing (historical) data and the output of a simulation
- why? using the (calibrated) model to predict the future

- Most simplest form: minimize mean square error between observed data points and simulated data points

Example Applications:

- weather/traffic forecasting
- well-drilling in oil industry
- trading

Coffee Tasting Problem

- Find a mixture of coffee in order to keep the coffee taste from one year to another
- Objective function = opinion of one expert

PUBLIC DOMAIN

BY SA
Quasipalm

M. Herdy: "Evolution Strategies with subjective selection", 1996

Many Problems, Many Algorithms?

Observation:

- Many problems with different properties
- For each, it seems a different algorithm?

In Practice:

- often most important to categorize your problem first in order to find / develop the right method
- → problem types

Algorithm design is an art,
what is needed is skill, intuition, luck, experience,
special knowledge and craft

freely translated and adapted from Ingo Wegener (1950-2008)

Problem Types

- discrete vs. continuous
  - discrete: integer (linear) programming vs. combinatorial problems
  - continuous: linear, quadratic, smooth/nonsmooth, blackbox/DFO, ...
  - both discrete&continuous variables: mixed integer problem
- constrained vs. unconstrained

Not (or only slightly) covered in this introductory lecture:

- deterministic vs. stochastic
- one or multiple objective functions

General Concepts in Optimization

- search domain
  - discrete, continuous variables
  - finite vs. infinite dimension
- constraints
  - bounds
  - linear/quadratic/non-linear constraint
  - blackbox constraint

Further important aspects (in practice):

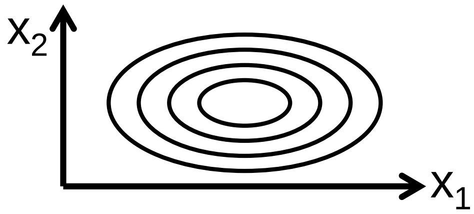
- deterministic vs. stochastic algorithms
- exact vs. approximation algorithms vs. heuristics
- anytime algorithms
- simulation-based optimization problem / expensive problem

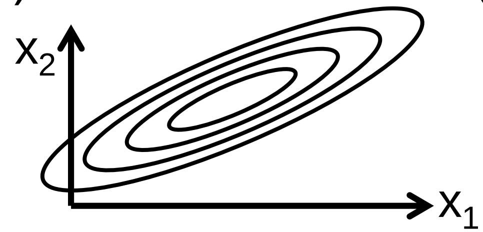
Problems and Instances

A *problem* is a general concept, what needs actually to be solved is an *instance*.

Examples:

- Knapsack Problem:
  - the general formulation of slide 14 defines the problem
  - an instance is given by the assignment of weights and profits to n items and by fixing the knapsack size W
- Convex-quadratic Functions: $f(x) = a^T x + ½\, x^T B x$
  - continuous problem with ellipsoidal level sets / lines of equal function value where B is symmetric, positive, and semi-definite
  - an instance is given by a specific rotation of the standard ellipses, their shapes (both via 'B') and their center (via 'a')





Excursion: The O-Notation

Excursion: The O-Notation

Motivation:

- we often want to characterize how quickly a function f(x) grows asymptotically
- e.g. when we say an algorithm takes n^2 steps to find the optimum of a problem with n (binary) variables, it is never exactly n^2, but maybe n^2+1 or $(n+1)^2$

Big-O Notation

should be known, here mainly restating the definition:

Definition 1 *We write $f(x) = O(g(x))$ iff there exists a constant $c > 0$ and an $x_0 > 0$ such that $f(x) \leq c|g(x)|$ holds for all $x > x_0$.*

we also view O(g(x)) as the set of all functions growing at most as quickly as g(x) and write $f(x) \in O(g(x))$

Big-O: Examples

- $f(x) + c = O(f(x))$ [as long as f(x) does not converge to zero]
- $c \cdot f(x) = O(f(x))$
- $f(x) \cdot g(x) = O(f(x) \cdot g(x))$
- $3n^4 + n^2 - 7 = O(n^4)$

Intuition of the Big-O:

- if $f(x) = O(g(x))$ then g(x) gives an upper bound (asymptotically) for f
- constants don't play a role
- with Big-O, you should have '≤' in mind

Excursion: The O-Notation

Further definitions to generalize from ‘≤’ to ‘≥‘ and ‘=‘:

- $f(x) = \Omega(g(x))$ if $g(x) = O(f(x))$
- $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and $g(x) = O(f(x))$

Note: Definitions equivalent to ‘<‘ and ‘>’ exist as well, but are not needed in this course

Exercise O-Notation

Please order the following functions in terms of their asymptotic behavior (from smallest to largest):

- $\exp(n^2)$
- $\log n$
- $\ln n / \ln \ln n$
- n
- $n \log n$
- $\exp(n)$
- $\ln(n!)$

Give for three of the relations a formal proof.

Exercise O-Notation (Solution)

Correct ordering:

$\frac{\ln(n)}{\ln(\ln(n))} = O(\log n)$ log n = O(n) n = O(n log n)

n log n = Θ(ln(n!)) ln(n!)= O(e^n) e^n = O($e^{n\text{^}2}$)

but for example $e^{n\text{^}2} \neq O(e^n)$

One exemplary proof:

$\frac{\ln(n)}{\ln(\ln(n))} = O(\log n)$:

$$\frac{\ln(n)}{\ln(\ln(n))} = \frac{\log(n)}{\log(e)\ln(\ln(n))} \underset{\text{for } n > 1}{\leq} \frac{3\log(n)}{\ln(\ln(n))} \underset{\text{for } n > 15}{\leq} 3|\log(n)|$$

Exercise O-Notation (Solution)

One more proof: ln n! = O(n log n)

- Stirling's approximation: $n! \sim \sqrt{2\pi n}\,(n/e)^n$ or even

  $$\sqrt{2\pi}\, n^{n+1/2} e^{-n} \leq n! \leq e\, n^{n+1/2} e^{-n}$$

- $\ln n! \leq \ln(e n^{n+\frac{1}{2}} e^{-n}) = 1 + \left(n + \frac{1}{2}\right) \ln n - n$

  $$\leq \left(n + \frac{1}{2}\right) \ln n \leq 2n \ln n = 2n\,\frac{\log n}{\log e} = c \cdot n \log n$$

  okay for $c = 2/\log e$ and all $n \in \mathbb{N}$

- n ln n = O(ln n!) proven in a similar vein

Introduction to Discrete Optimization

Discrete Optimization

Discrete optimization:

- discrete variables
- or optimization over discrete structures (e.g. graphs)
- search space often finite, but typically too large for enumeration
- → need for smart algorithms

Algorithms for discrete problems:

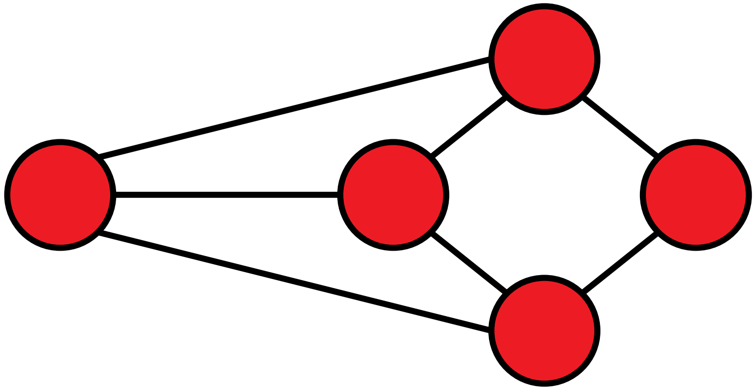
- typically problem-specific
- but some general concepts are repeatedly used:
  - greedy algorithms (lecture 2)
  - branch&bound (lecture 3)
  - dynamic programming (lecture 4)
  - heuristics (lecture 5)

Basic Concepts of Graph Theory

[following for example http://math.tut.fi/~ruohonen/GT_English.pdf]

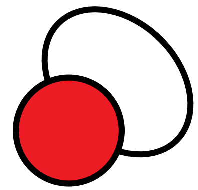
Graphs

Definition 1 *An undirected graph G is a tupel $G = (V, E)$ of edges $e = \{u, v\} \in E$ over the vertex set V (i.e., $u, v \in V$).*



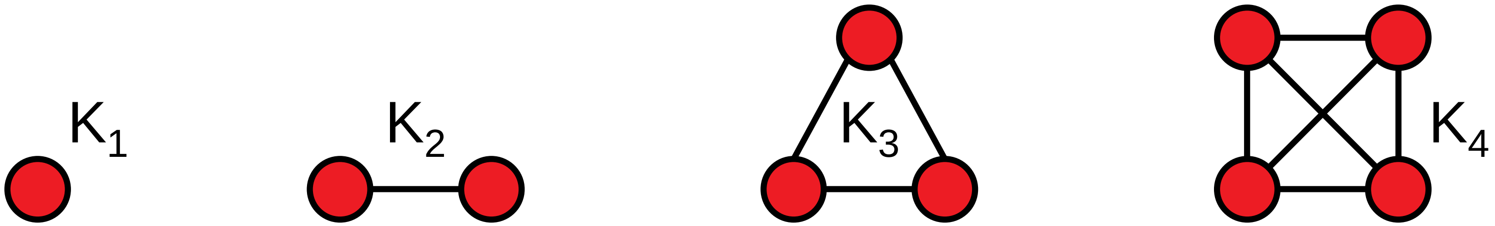
- vertices = nodes
- edges = lines
- Note: edges cover two *unordered* vertices (*undirected* graph)
  - if they are *ordered*, we call G a *directed* graph

Graphs: Basic Definitions



a loop

- G is called *empty* if E empty
- u and v are *end vertices* of an edge {u,v}
- Edges are *adjacent* if they share an end vertex
- Vertices u and v are *adjacent* if {u,v} is in E
- The *degree* of a vertex is the number of times it is an end vertex
- A complete graph contains all possible edges (once):

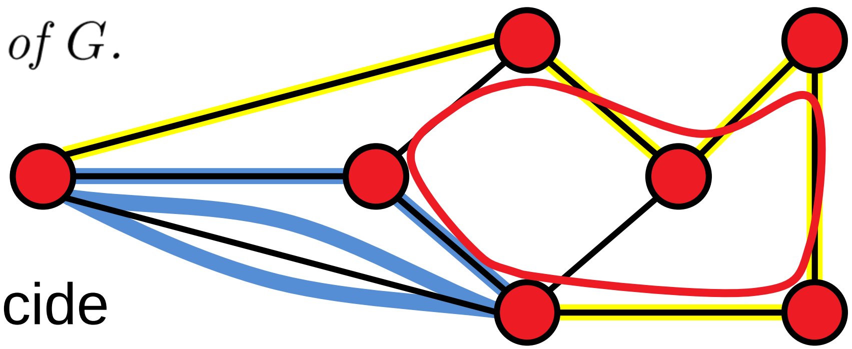


Walks, Paths, and Circuits

Definition 1 *A* walk *in a graph* $G = (V, E)$ *is a sequence*

$$v_{i_0}, e_{i_1} = (v_{i_0}, v_{i_1}), v_{i_1}, e_{i_2} = (v_{i_1}, v_{i_2}), \ldots, e_{i_k}, v_{i_k},$$

alternating vertices and adjacent edges of G.



A walk is

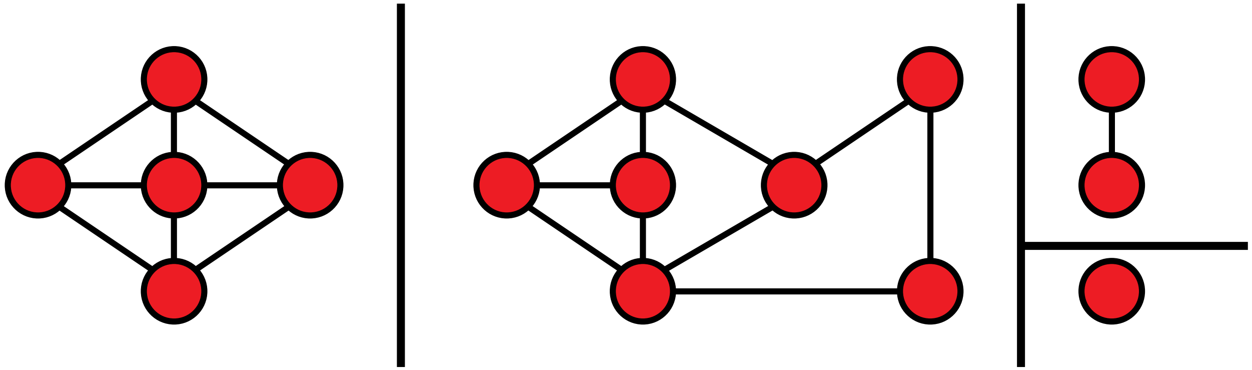
- *closed* if first and last node coincide
- a *trail* if each edge traversed at most once
- a *path* if each vertex is visited at most once

- a closed path is a *circuit* or *cycle*
- a closed path involving all vertices of G is a *Hamiltonian cycle*

Graphs: Connectedness

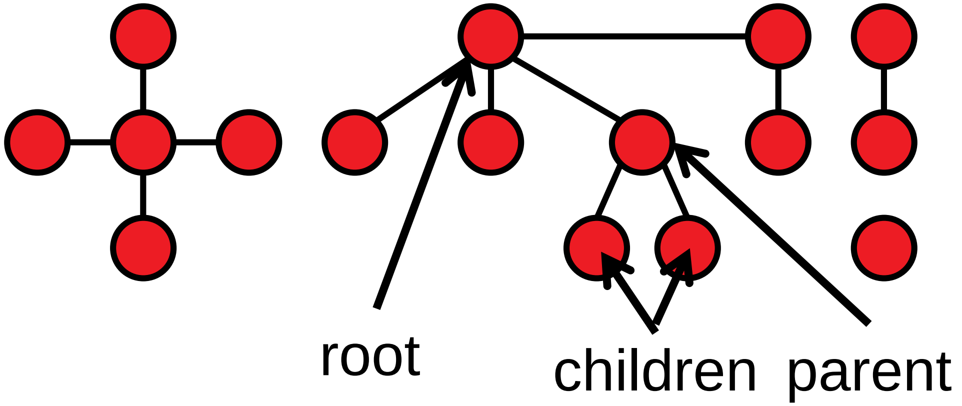
- Two vertices are called *connected* if there is a walk between them in G
- If all vertex pairs in G are connected, G is called connected

- The *connected components* of G are the (maximal) subgraphs which are connected.

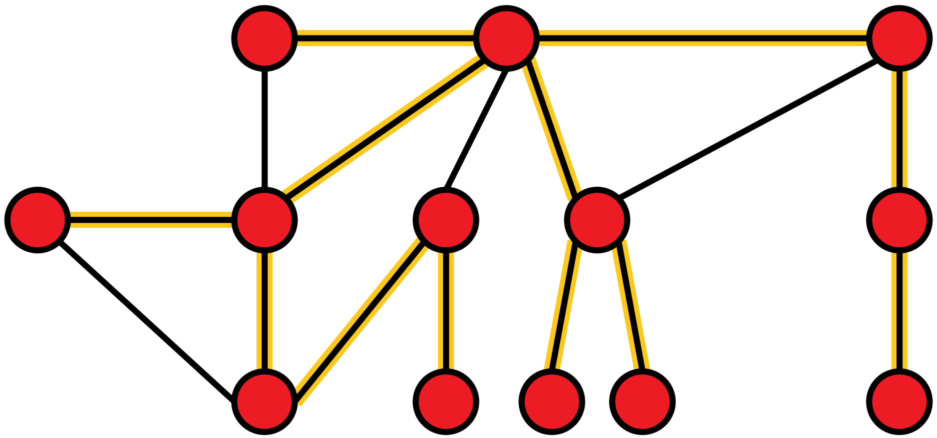


Trees and Forests

- A *forest* is a cycle-free graph
- A *tree* is a connected forest



A *spanning tree* of a connected graph G is a tree in G which contains all vertices of G



Depth-First Search (DFS)

Sometimes, we need to traverse a graph, e.g. to find certain vertices

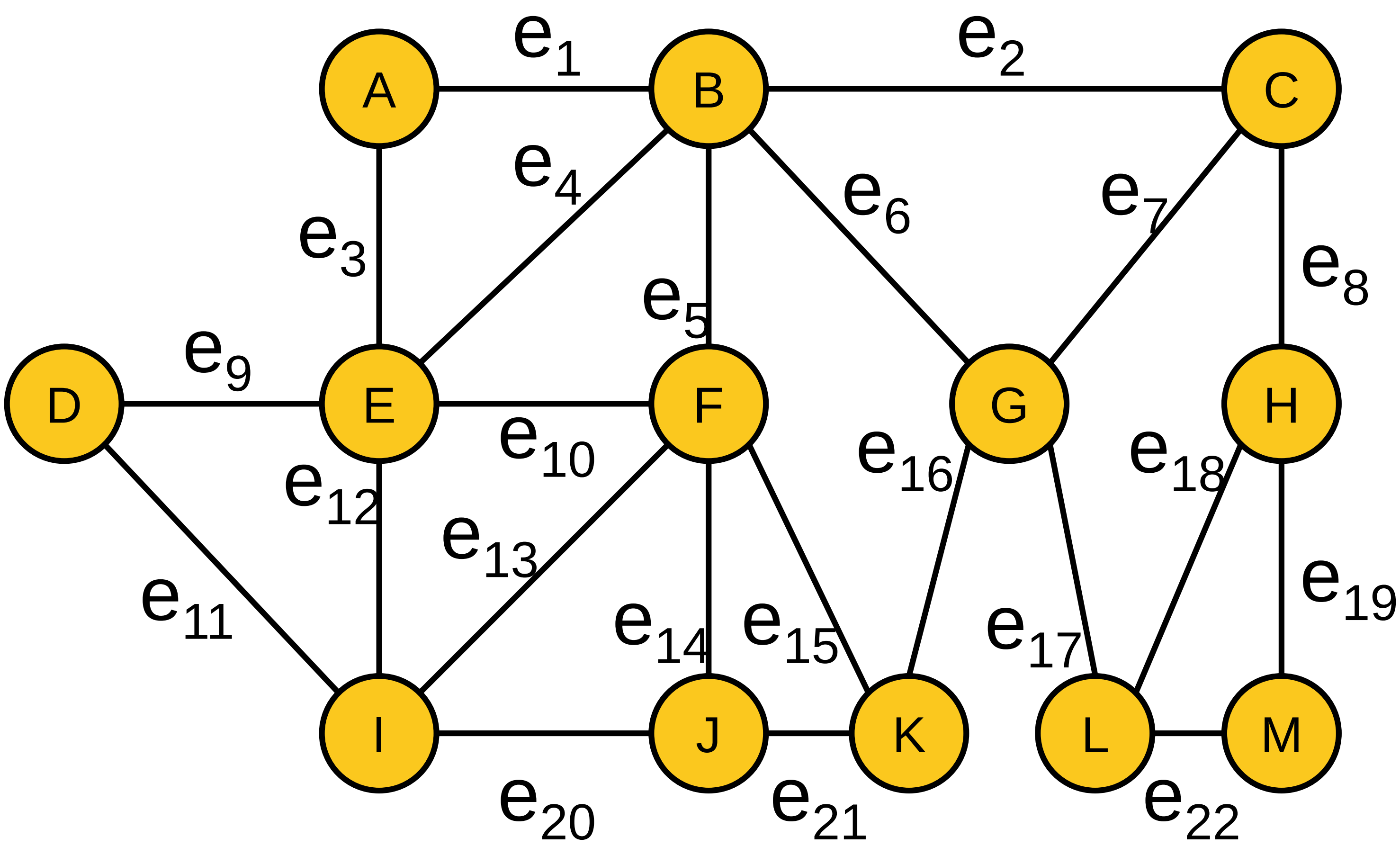
Depth-first search and breadth-first search are two algorithms to do so

Depth-first Search (for undirected/acyclic and connected graphs)

1. start at any node x; set i=0
2. as long as there are unvisited edges {x,y}:
   - choose the next unvisited edge {x,y} to a vertex y and mark x as the parent of y
   - if y has not been visited so far: i=i+1, give y the number i, and continue the search at x=y in step 2
   - else continue with next unvisited edge of x
3. if all edges {x,y} are visited, we continue with x=parent(x) at step 2 or stop if x==v0

DFS: Stage Exercise

Exercise the DFS algorithm on the following graph!



Breadth-First Search (BFS)

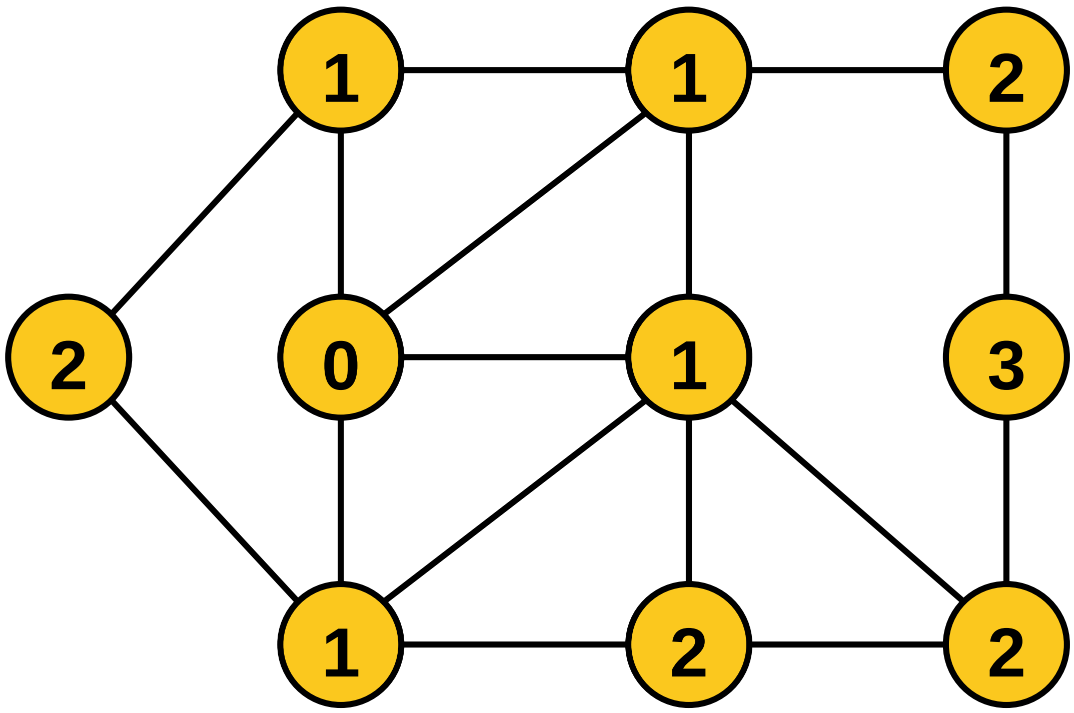
Breadth-first Search (for undirected/acyclic and connected graphs)

❶ start at any node x, set i=0, and label x with value i

❷ as long as there are unvisited edges {x,y} which are adjacent to a vertex x that is labeled with value i:

- label all vertices y with value i+1

❸ set i=i+1 and go to step 2

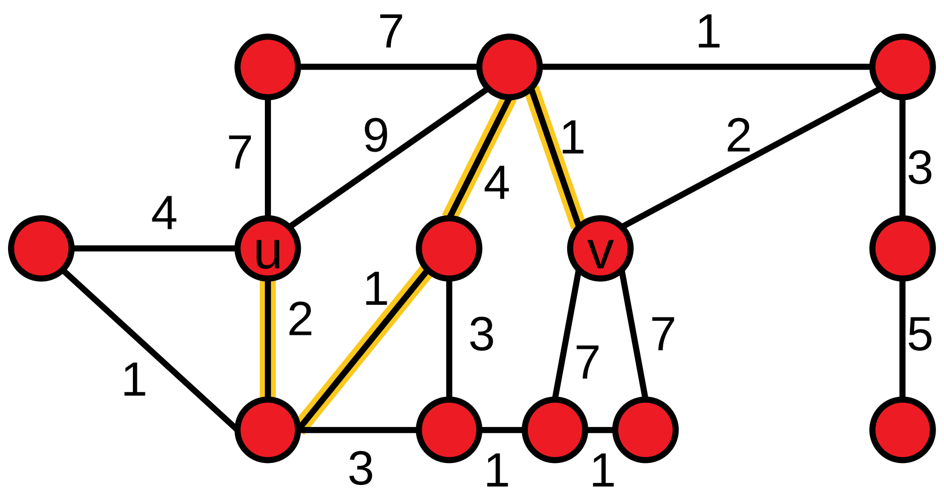


Definition of Some Combinatorial Problems Used Later on in the Lecture

Shortest Paths (SP)

Shortest Path problem:

Given a graph G=(V,E) with edge weights w_i for each edge e_i.
Find the shortest path from a vertex v to a vertex u, i.e., the path $(v, e_1=\{v, v_1\}, v_1, ..., v_k, e_k=\{v_k,u\}, u)$ such that $w_1 + ... + w_k$ is minimized.



Obvious Applications

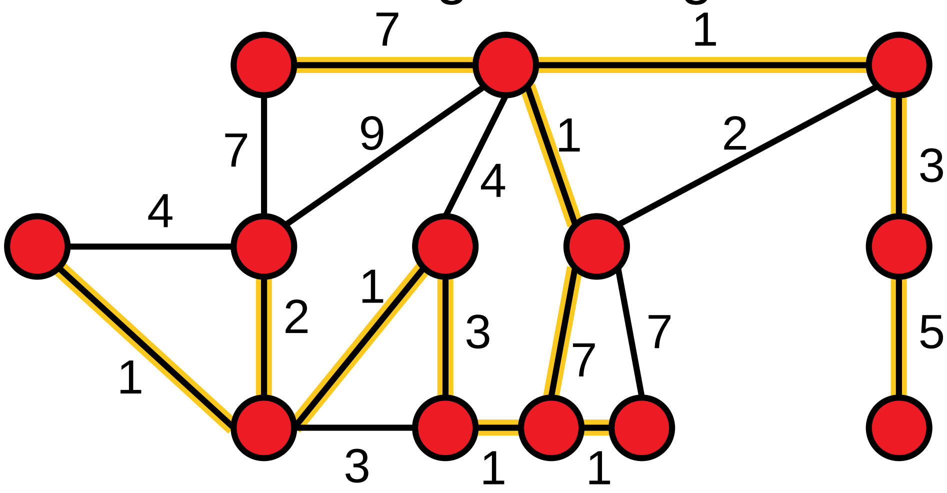
Google maps

Finding routes for packages in a computer network

...

Minimum Spanning Tree problem:

Given a graph G=(V,E) with edge weights w_i for each edge e_i.
Find the spanning tree with the smallest weight among all spanning trees.



Applications

Setting up a new wired telecommunication/water supply/electricity network

Constructing minimal delay trees for broadcasting in networks

Set Cover Problem (SCP)

Set Cover Problem

Given a set U={1, 2, 3, ..., n}, called the universe, and a set S={s_1, ..., s_m} of m subsets of U, the union of which equals U. Find the smallest subset of S, the union of which also equals U. In other words, find an index $I \subseteq \{1, ..., m\}$ which minimizes $\Sigma_{i \in I} |s_i|$ such that the union of the s_i ($i \in I$) equals U.

U = {1,2,3,4,5}
S = {{1,2}, {1,3,5}, {1,2,3,5}, {2,3,4}}

minimal set cover: {1,3,5} {2,3,4}

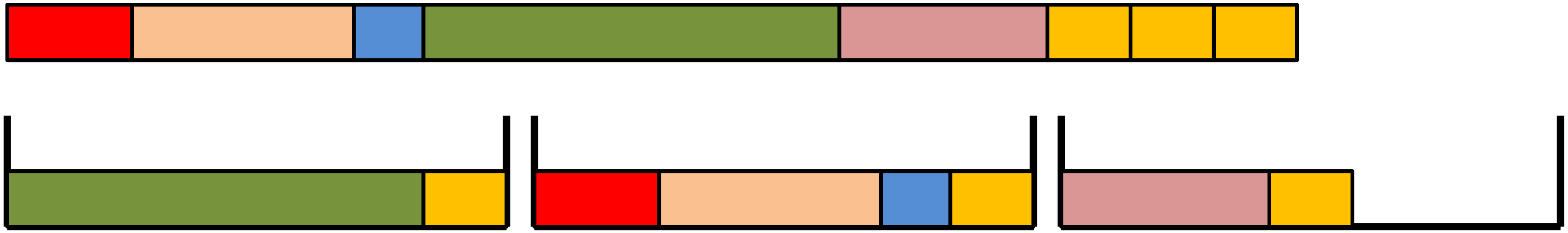
Application example

IBM's Antivirus use(d) set cover to search for a minimal set of code snippets which appear in all known viruses but not in "good" code

Bin Packing (BP)

Bin Packing Problem

Given a set of n items with sizes a_1, a_2, ..., a_n. Find an assignment of the a_i's to bins of size V such that the number of bins is minimal and the sum of the sizes of all items assigned to each bin is $\leq$ V.



Applications

similar to multiprocessor scheduling of n jobs to m processors

Satisfiability Problem (SAT)

Notations:

- A *Boolean expression* is built from literals, operators and parentheses.
- A *literal* is either a Boolean variable x_i or its negation $\overline{x_i}$
- *Operators* are AND (conjunction), OR (disjunction), and NOT (negation)
- A formula is *satisfiable* if there is an assignment (TRUE/FALSE) to each of the variables that makes the whole formula TRUE

The Boolean satisfiability problem (SAT):

Given a Boolean expression E, is E satisfiable?

Satisfiability Problem (SAT)

The Boolean satisfiability problem (SAT):

Given a Boolean expression E, is E satisfiable?

Example:

$(x_1 \text{ OR } \overline{x_2}) \text{ AND } (\overline{x_1} \text{ OR } x_2 \text{ OR } \overline{x_3}) \text{ AND } (\overline{x_1} \text{ OR } \overline{x_4}) \text{ AND } (x_3 \text{ OR } x_4)$

Possible truth assignment: x_1=TRUE, x_2=TRUE, x_3=TRUE, x_4=FALSE

Applications:

- many, ranging from formal verification over artificial intelligence to machine learning and data mining
- examples: equivalence checking of Boolean circuits, automated test pattern generation, AI planning

Integer Linear Programming (ILP)

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ \text{and} \quad & x \in \mathbb{Z}^n \end{aligned}$$

- rather a problem class
- can be written as ILP: SAT, TSP, Vertex Cover, Set Packing, ...

Conclusions I

- many, many more problems out there
- typically in practice: need to solve very specific instances
- here only possible to provide you
  - the basic algorithm design ideas
  - applied to a few standard problem classes
  - regular training (i.e. exercises) to gain intuition and experience
  - a broad overview on optimization topics to potentially draw your interest (e.g. towards a PhD on that topic)

Conclusions II

I hope it became clear...

...what optimization is about

...what is a graph, a node/vertex, an edge, ...

...and that designing a good algorithm is an important task