Introduction to Optimization

September 18, 2015

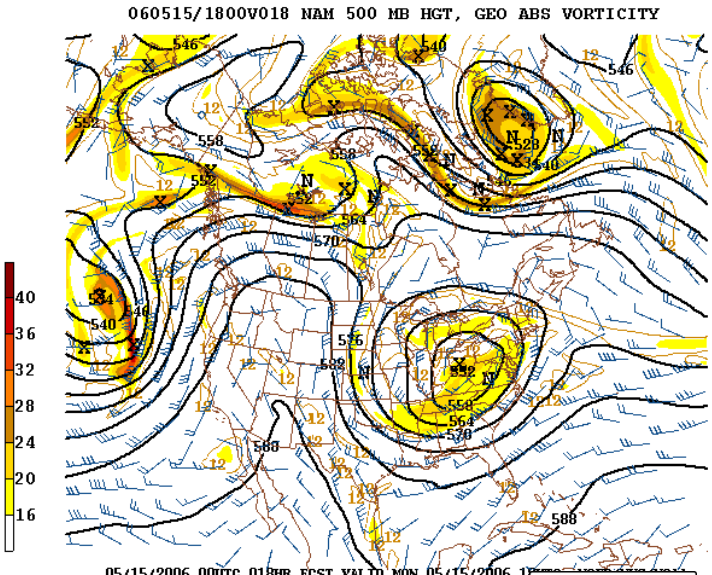
TC2 - Optimisation

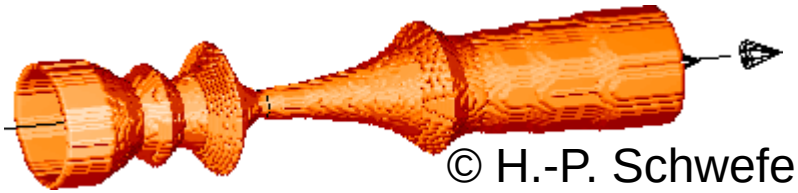
Université Paris-Saclay, Orsay, France

Anne Auger
INRIA Saclay – Ile-de-France

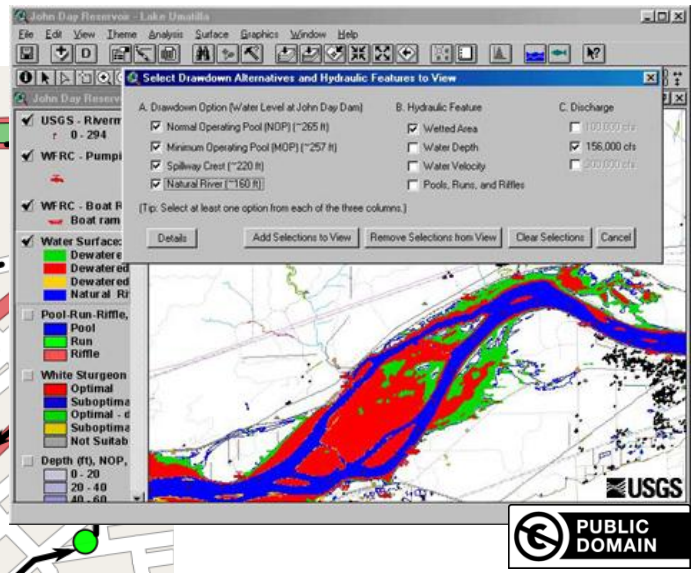
Dimo Brockhoff
INRIA Lille – Nord Europe

What is Optimization?

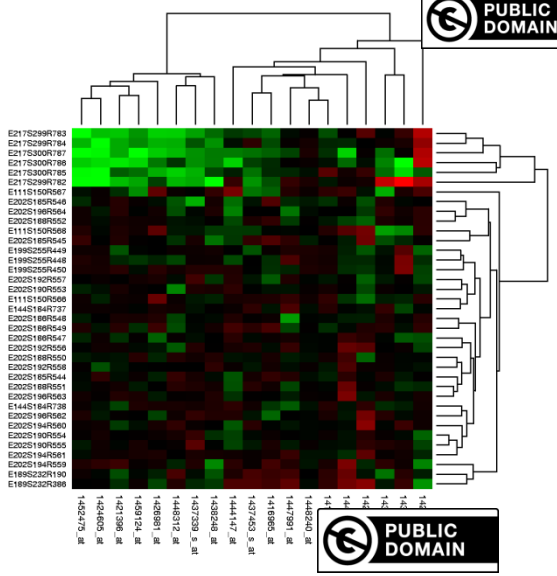




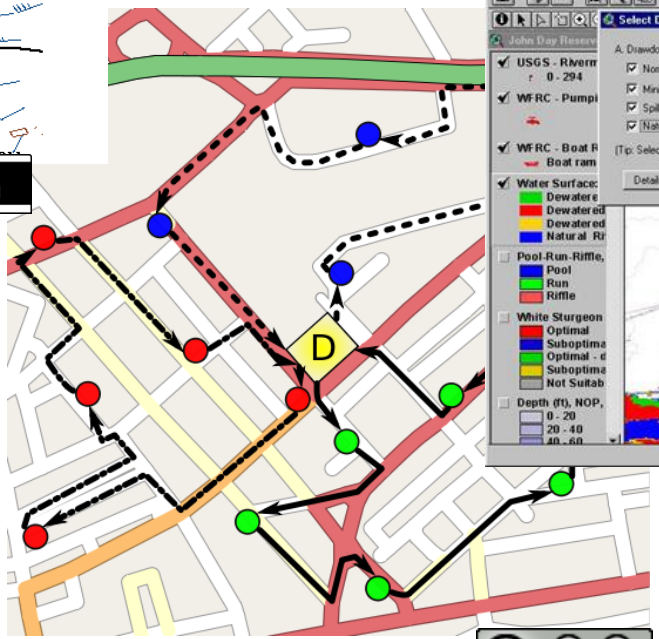
© H.-P. Schwefel



PUBLIC DOMAIN



PUBLIC DOMAIN



Maly LOLek

CC BY SA

What is Optimization?

- find solutions x which minimize f(x) in the shortest time possible (maximization is reformulated as minimization) or
- find solutions x with as small f(x) 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 clique of size 5 in a graph!"

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

Course Overview

Date		Topic
Fri, 18.9.2015	DB	Introduction and Greedy Algorithms
Fri, 25.9.2015	DB	Dynamic programming and Branch and Bound
Fri, 2.10.2015	DB	Approximation Algorithms and Heuristics
Fri, 9.10.2015	AA	Introduction to Continuous Optimization
Fri, 16.10.2015	AA	End of Intro to Cont. Opt. + Gradient-Based Algorithms I
Fri, 30.10.2015	AA	Gradient-Based Algorithms II
Fri, 6.11.2015	AA	Stochastic Algorithms and Derivative-free Optimization
16 - 20.11.2015		Exam (exact date to be confirmed)

all classes + exam are from 14h till 17h15 (incl. a 15min break)
here in PUIO-D101/D103

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:

1. give a broad overview of where and how optimization is used
2. understand the fundamental concepts of optimization algorithms
3. be able to apply common optimization algorithms on real-life (engineering) problems

there will be also an optional class "Blackbox Optimization" which we will present briefly in next week's class

The Exam

- 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)
- date to be confirmed soon, but within November 16–20, 2015

- counts 2/3 of overall grade

Mid-term Exam (aka “contrôle continu”)

- we will have one larger home exercise
- hand-out ready by next Friday
- to be solved at home in addition to the lecture
- hand-in by e-mail at a specific deadline (to be announced next week, most likely in mid October)
- graded: need 50% to pass, counts as 1/3 of overall grade

All information also available at

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

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

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
  - greedy algorithms applied to the money change and the minimum spanning tree problem

General Context Optimization

Given:

set of possible solutions — *Search space*

quality criterion — *Objective function*

Objective:

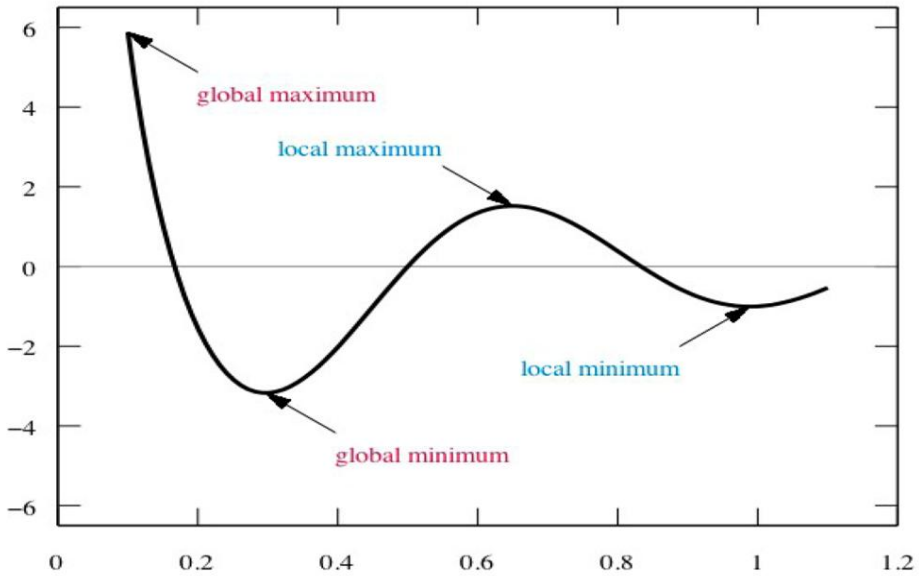
Find the best possible solution for the given criterion

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]

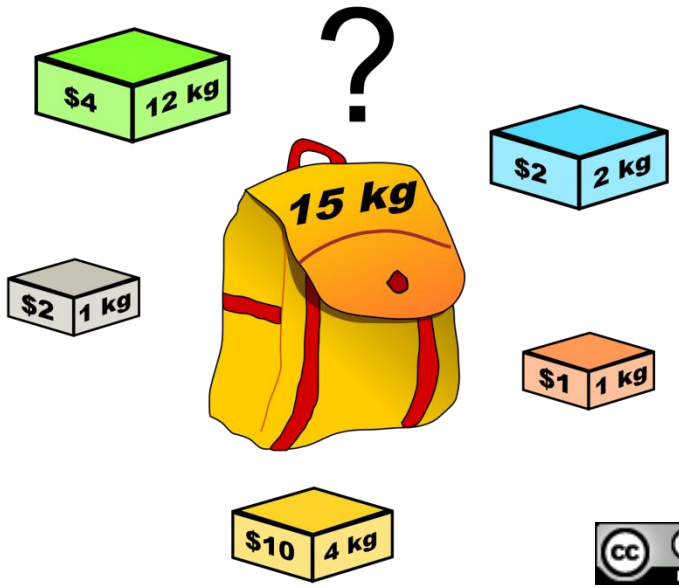
Hard constraints *must* be satisfied while soft constraints are preferred to hold but are not required to be satisfied

[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



Dake

[Problem of ressource allocation with financial constraints]

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

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

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

Example 2: Combinatorial Optimization

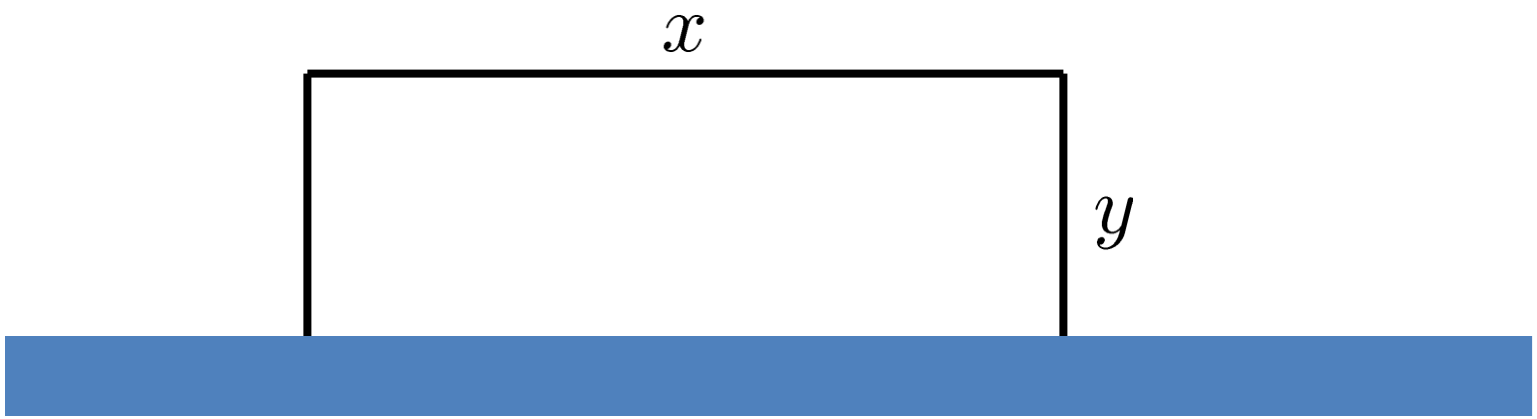
Traveling Salesperson Problem (TSP)

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

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

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?



solution can be found analytically:

exercise for the weekend ;-)

$$\Omega = \mathbb{R}^2 :$$
$$\max xy$$
$$\text{where } x + 2y \leq 500$$

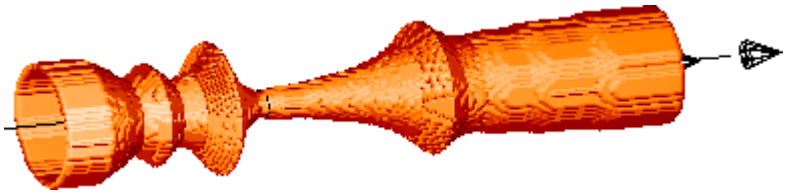
Example 4: A "Manual" Engineering Problem

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

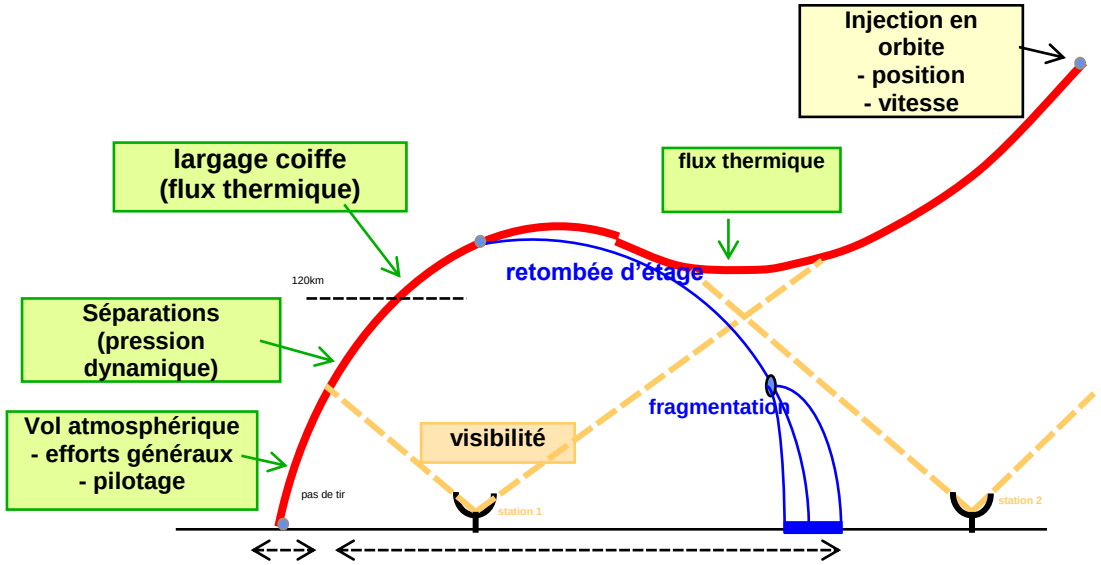
copyright Hans-Paul Schwefel

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

Example 5: Constrained Continuous Optimization

Design of a Launcher

Poppy



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

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

Example 6: History Matching/Parameter Calibration

One wide class of problems:

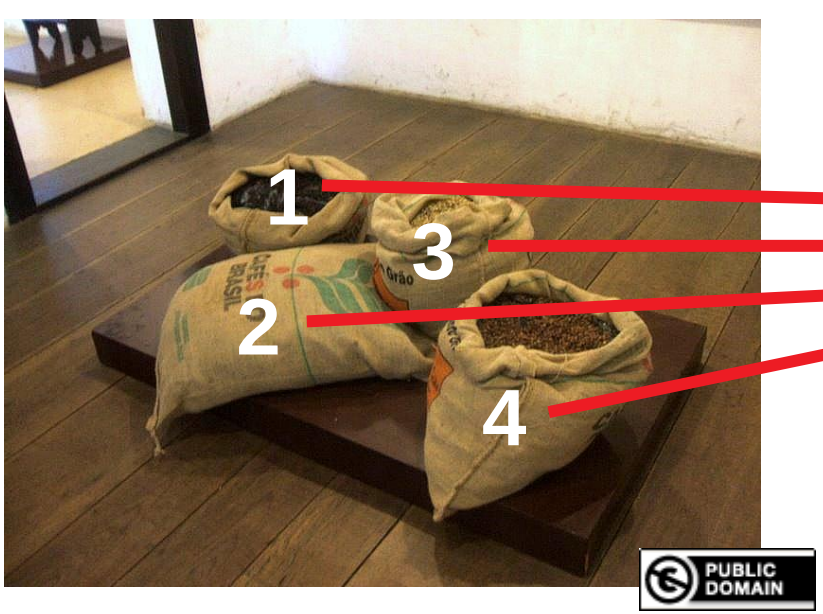
- matching existing (historical) data and the output of a simulation
- why? using the (calibrated) model to predict the future

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

Example 7: Interactive Optimization

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



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 covered in this introductory lecture:

- deterministic vs. stochastic
- one or multiple objective functions

General Concepts in Optimization

- search domain
  - discrete vs. continuous variables vs. mixed integer
  - 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

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 quadratically many steps (in the input size) to find the optimum of a problem with n (binary) variables, it is most likely not 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 a set of functions growing at most as quick as g(x) and write $f(x) \in O(g(x))$

Big-O: Examples

- $f(x) + c = O(f(x))$ [if $f(x)$ does not go to zero for x to infinity]
- $c \cdot f(x) = O(f(x))$
- $f(x) \cdot g(x) = O(f(x) \cdot g(x))$
- $3n^4 + n^2 - 7 = O(n^4)$

Intuition of the Big-O:

- if $f(x) = O(g(x))$ then $g(x)$ gives an upper bound (asymptotically) for f
- With Big-O, you should have '$\leq$' 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 $g(x) = O(f(x))$

Definition 2 *We write $f(x) = o(g(x))$ iff for every constant $\epsilon > 0$ there exists an $x_0 > 0$ such that $f(x) \leq \epsilon |g(x)|$ holds for all $x > x_0$.*

Note that “$f(x) = o(g(x))$” is equivalent to “$\lim_{x \to \infty} f(x)/g(x) = 0$” as long as $g(x)$ is nonzero after an x_0

- $f(x) = \omega(g(x))$ if $g(x) = o(f(x))$

only proving upper bounds to compare algorithms is not sufficient!

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:

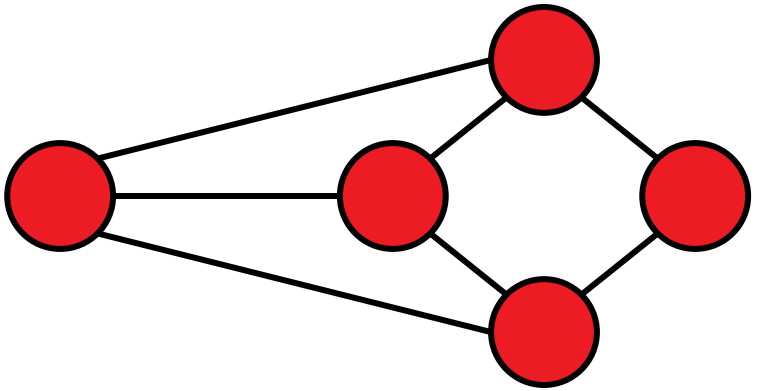
- typically problem-specific
- but some general concepts repeatably used:
  - greedy algorithms (today)
  - dynamic programming (next week)
  - branch&bound (next week)
  - heuristics (lecture 3)

Basic Concepts of Graph Theory

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

Graphs

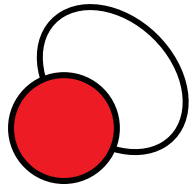
Definition 1 *An undirected graph G is a tupel $G = (V, E)$ of edges $e = \{u, v\} \in E$ over the vertex set V (i.e., $u, v \in V$).*



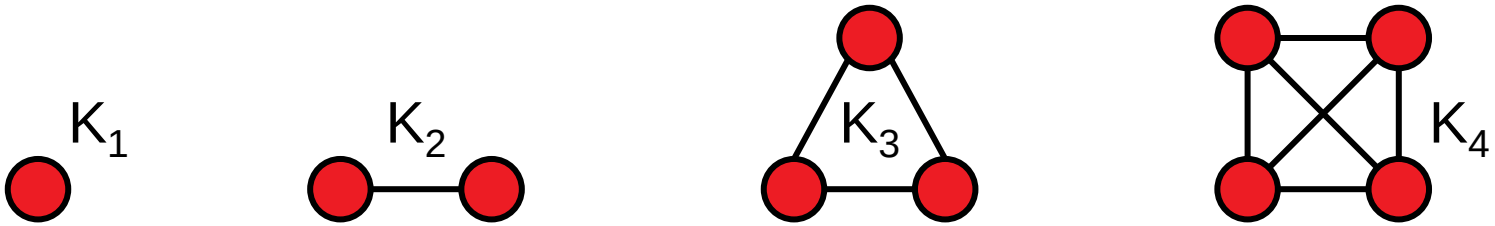
- vertices = nodes
- edges = lines
- Note: edges cover two *unordered* vertices (*undirected* graph)
  - if they are *ordered*, we call G a *directed* graph

Graphs: Basic Definitions

- 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):



a loop

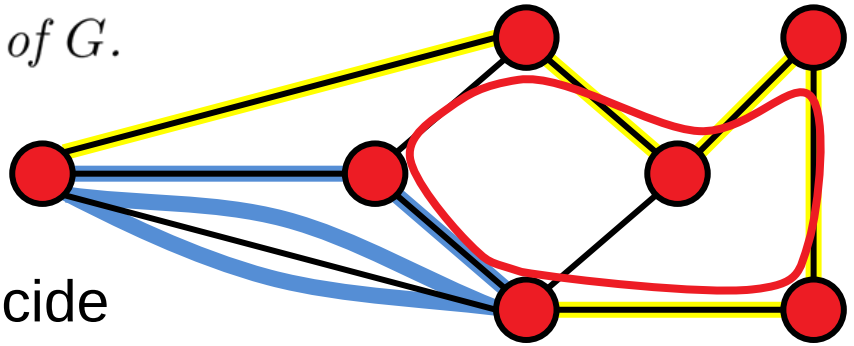


Walks, Paths, and Circuits

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

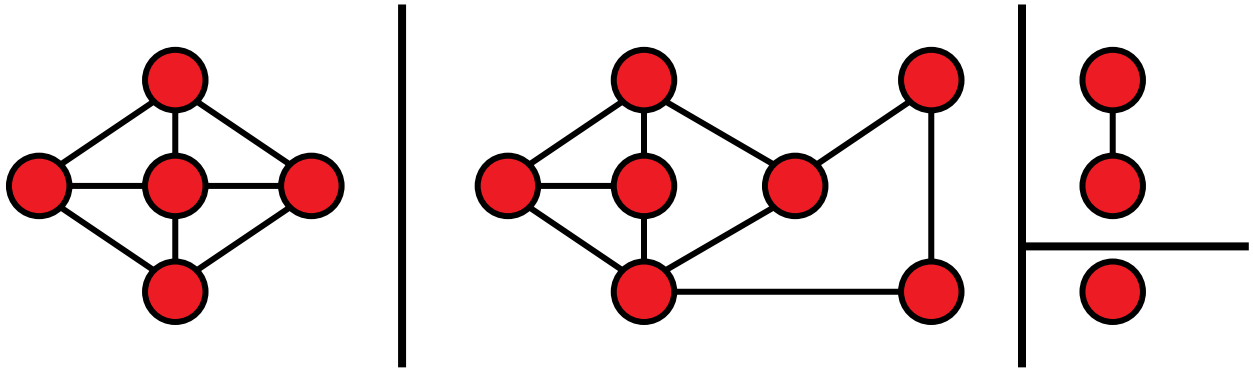
- *closed* if first and last node coincide
- a *trail* if each edge traversed at most once
- a *path* if each vertex is visited at most once

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

Graphs: Connectedness

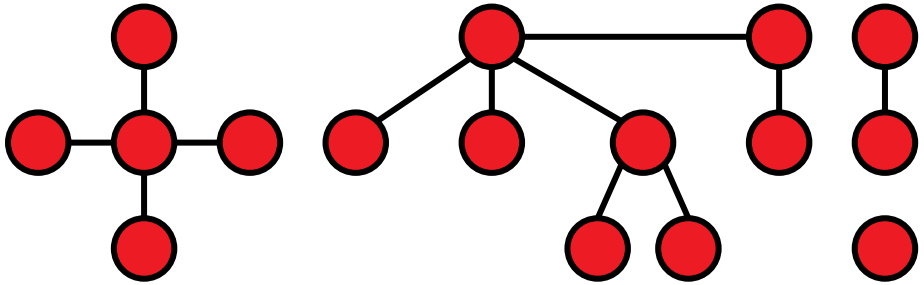
- Two vertices are called *connected* if there is a walk between them in G
- If all vertex pairs in G are connected, G is called connected

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

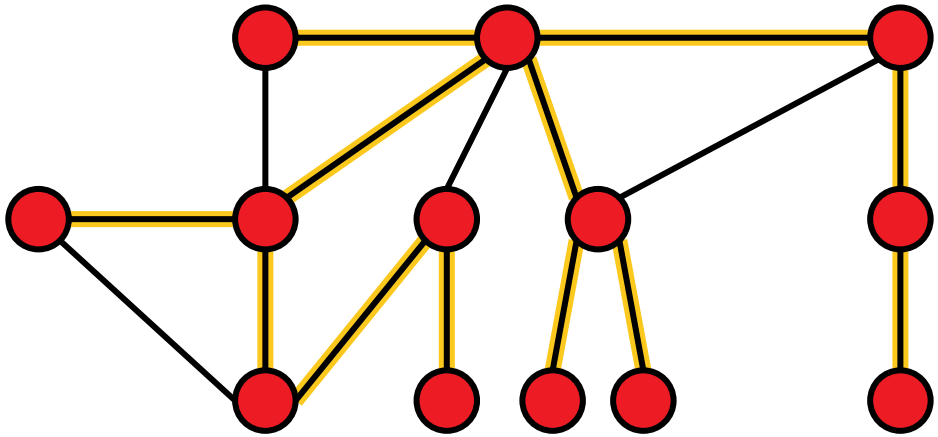


Trees and Forests

- 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



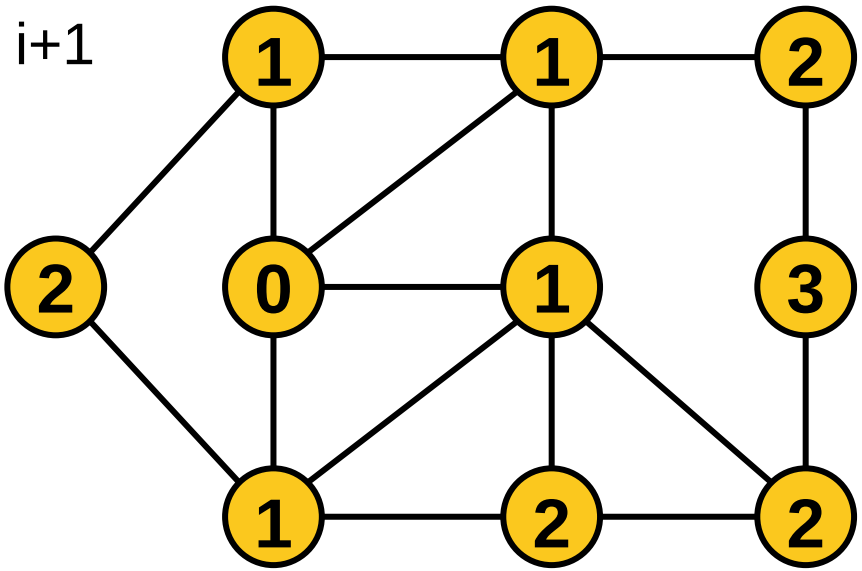
Breadth-First Search (BFS)

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 (here only BFS):

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

1. start at any node x, set i=0, and label x with value i
2. 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
3. set i=i+1 and go to step 2

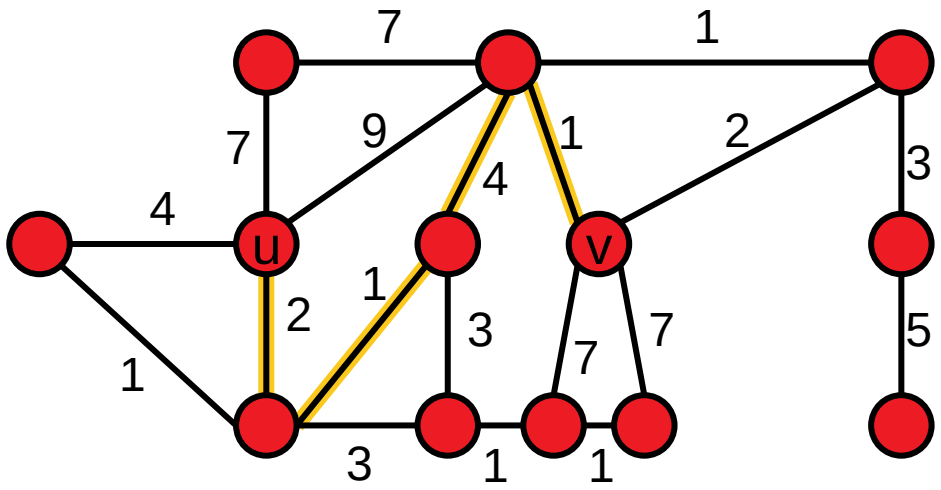


Definition of Some Combinatorial Problems Used Later on in the Lecture

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

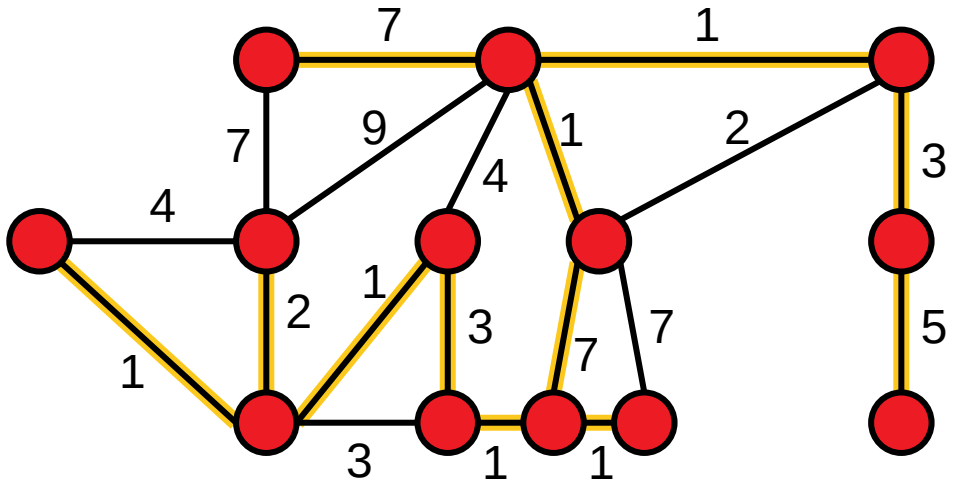
Finding routes for packages in a computer network

...

Minimum Spanning Trees (MST)

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

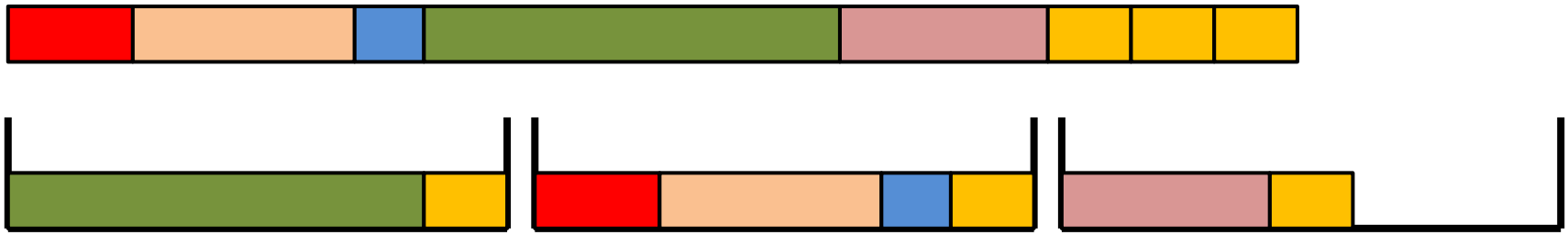
Setting up a new wired telecommunication/water supply/electricity network

Constructing minimal delay trees for broadcasting in networks

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

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, ...
- interesting relation between the algorithm for the continuous case and integer solutions: if A is totally unimodular and b integer, the ILP has an integer solution

Preliminary 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)

Preliminary Conclusions II

I hope that, so far, 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

Greedy Algorithms

From Wikipedia:

> "A *greedy algorithm* is an algorithm that follows the problem solving *heuristic* of making the locally optimal choice at each stage with the hope of finding a global optimum."

- Note: typically greedy algorithms do not find the global optimum
- We will see later when this is the case

Greedy Algorithms: Overview

- Example 1: Money Change
- Example 2: Minimal Spanning Trees (MST) and the algorithm of Kruskal

Example 1: Money Change

Change-making problem

- Given n coins of distinct values $w_1=1$, w_2, ..., w_n and a total change W (where w_1, ..., w_n, and W are integers).
- Minimize the total amount of coins Σx_i such that $\Sigma w_i x_i = W$ and where x_i is the number of times, coin i is given back as change.

Greedy Algorithm

Unless total change not reached:

add the largest coin which is not larger than the remaining amount to the change

Note: only optimal for standard coin sets, not for arbitrary ones!

Related Problem:

finishing darts (from 501 to 0 with 9 darts)

Example 2: Minimal Spanning Trees (MST)

Outline:

- reminder of problem definition
- Kruskal's algorithm
- analysis of its running time
- proof of its correctness

MST: Reminder of Problem Definition

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

Minimum Spanning Tree Problem (MST):

Given a (connected) graph G=(V,E) with edge weights w_i for each edge e_i. Find a spanning tree T that minimizes the weights of the contained edges, i.e. where

$$\sum_{e_i \text{ in } T} w_i$$

is minimized.

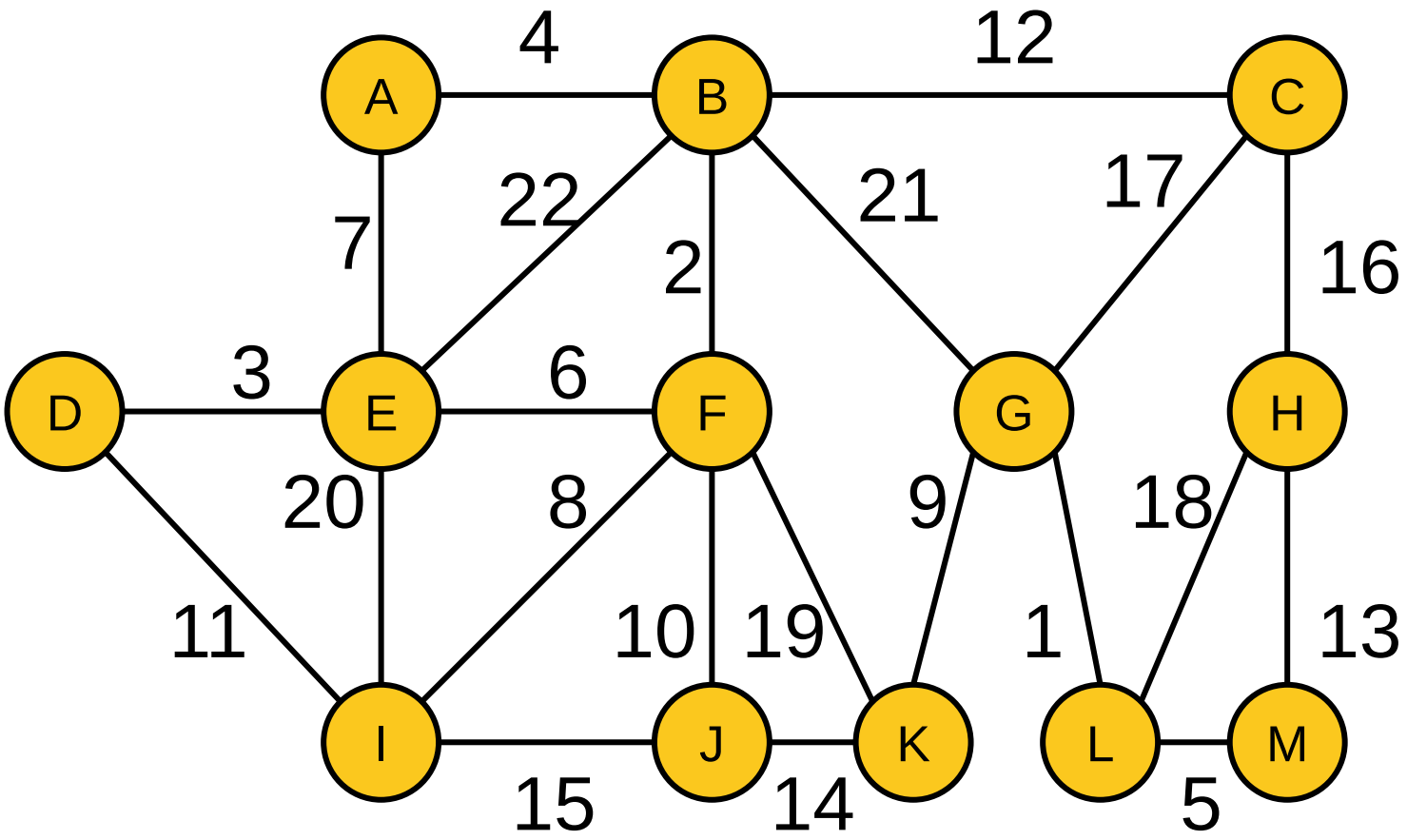
Kruskal's Algorithm: Idea

Algorithm, see [1]

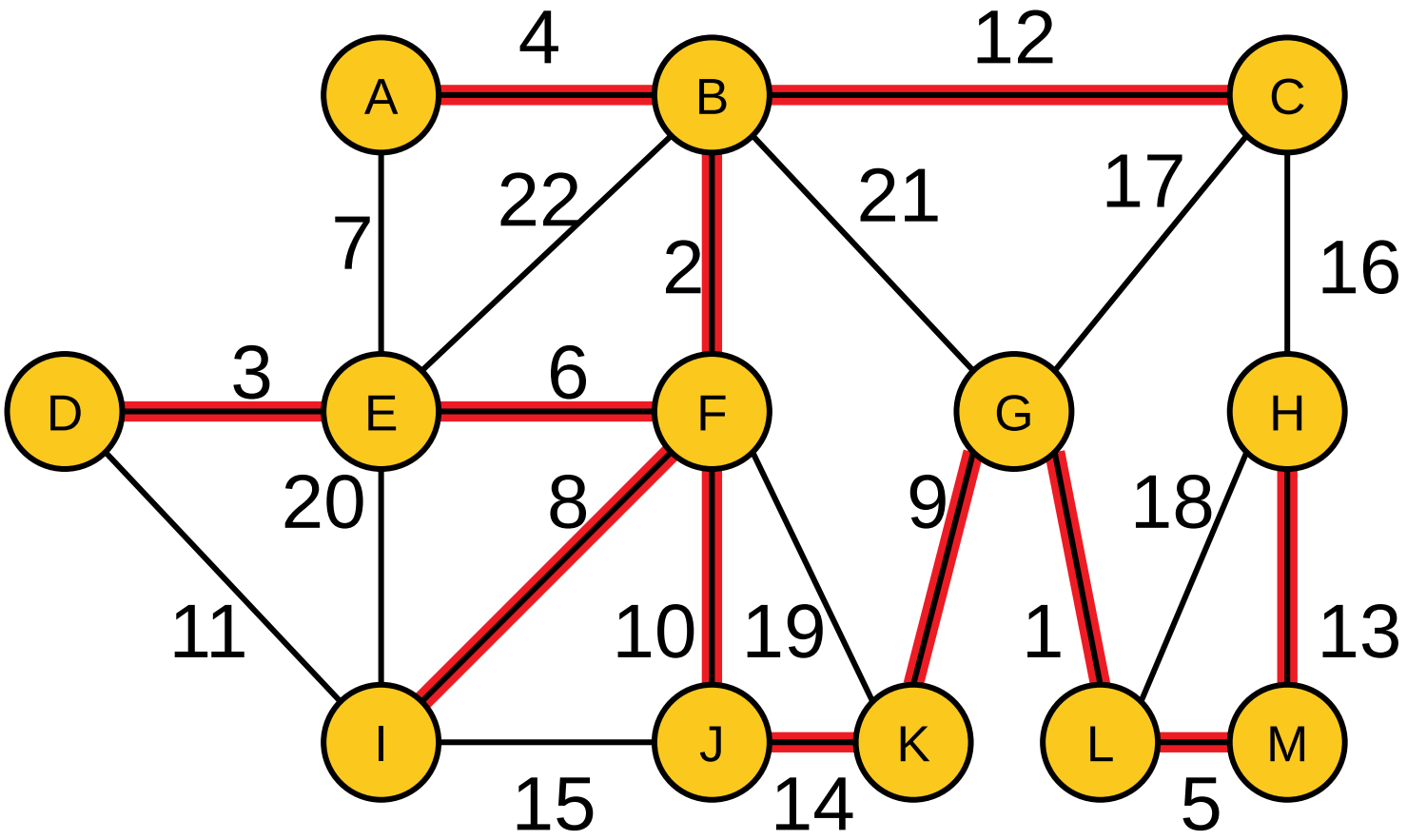
- Create forest F = (V,{}) with n components and no edge
- Put sorted edges (such that w.l.o.g. $w_1 < w_2 < ... < w_{|E|}$) into set S
- While S non-empty and F not spanning:
  - delete cheapest edge from S
  - add it to F if no cycle is introduced

[1] Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". *Proceedings of the American Mathematical Society* **7**: 48–50. doi:10.1090/S0002-9939-1956-0078686-7

Kruskal's Algorithm: Example



Kruskal's Algorithm: Example



Kruskal's Algorithm: Runtime Considerations

First question: how to implement the algorithm?

- sorting of edges needs O(|E| log |E|)

Algorithm

Create forest F = (V,{}) with n components and no edge

Put sorted edges (such that w.l.o.g. $w_1 < w_2 < ... < w_{|E|}$) into set S

While S non-empty and F not spanning:

delete cheapest edge from S

add it to F if no cycle is introduced

simple

?

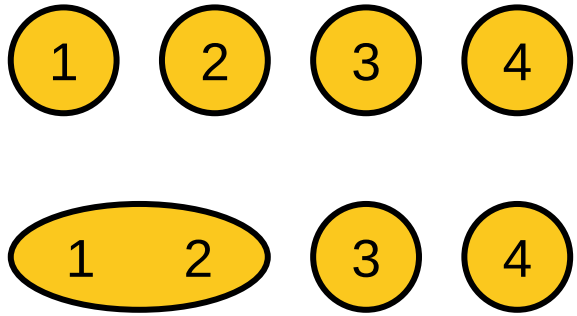
forest implementation:
Disjoint-set data structure

Disjoint-set Data Structure ("Union&Find")

Data structure: ground set 1...N grouped to disjoint sets

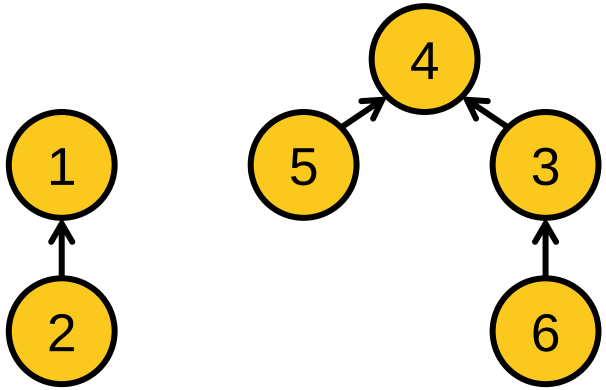
Operations:

- FIND(i): to which set does i belong?
- UNION(i,j): union the sets of i and j!



Implemented as trees:

- UNION(T1, T2): hang root node of smaller tree under root node of larger tree (constant time), thus
- FIND(u): traverse tree from u to root (to return a representative of u's set) takes logarithmic time in total number of nodes



Implementation of Kruskal's Algorithm

Algorithm, rewritten with UNION-FIND:

- Create initial disjoint-set data structure, i.e. for each vertex v_i, store v_i as representative of its set
- Create empty forest F = {}
- Sort edges such that w.l.o.g. $w_1 < w_2 < \ldots < w_{|E|}$
- for each edge e_i={u,v} starting from i=1:
  - if FIND(u) ≠ FIND(v): # no cycle introduced?
    - F = F ∪ {{u,v}}
    - UNION(u,v)
- return F

Back to Runtime Considerations

- Sorting of edges needs O(|E| log |E|)
- forest: **Disjoint-set data structure**
  - initialization: O(|V|)
  - log |V| to find out whether the minimum-cost edge {u,v} connects two sets (no cycle induced) or is within a set (cycle would be induced)
  - 2x FIND + potential UNION needs to be done O(|E|) times
  - total O(|E| log |V|)
- Overall: O(|E| log |E|)

Kruskal's Algorithm: Proof of Correctness

Two parts needed:

1. Algo always produces a spanning tree

   final F contains no cycle and is connected by definition ✓

2. Algo always produces a *minimum* spanning tree
   - argument by induction
   - P: If *F* is forest at a given stage of the algorithm, then there is some minimum spanning tree that contains *F*.
   - clearly true for F = (V, {})
   - assume that P holds when new edge e is added to F and be T a MST that contains F
     - if e in T, fine
     - if e not in T: T + e has cycle C with edge f in C but not in F (otherwise e would have introduced a cycle in F)
       - now T – f + e is a tree with same weight as T (since T is a MST and f was not chosen to F)
       - hence T – f + e is MST including T + e (i.e. P holds) ✓

Conclusion Greedy Algorithms I

What we have seen so far:

- two problems where a greedy algorithm was optimal
  - money change
  - minimum spanning tree (Kruskal's algorithm)
- but also: greedy not always optimal
  - in particular for NP-hard problems

Obvious Question: when is greedy good?

Answer: if the problem is a matroid (not covered here)

From Wikipedia: [...] a matroid is a structure that captures and generalizes the notion of linear independence in vector spaces. There are many equivalent ways to define a matroid, the most significant being in terms of independent sets, bases, circuits, closed sets or flats, closure operators, and rank functions.

Conclusions Greedy Algorithms II

I hope it became clear...

...what a greedy algorithm is

...that it not always results in the optimal solution

...but that it does if and only if the problem is a matroid