DM811
HEURISTICS AND LOCAL SEARCH ALGORITHMS FOR COMBINATORIAL OPTIMZATION

Lecture 2

Basics (continued)
Classical Techniques

Marco Chiarandini

Outline

Last Time

- Terminology: Combinatorial Problems
- Graph-vertex coloring
- Problem solving according Polya
- SAT problem
- Basic Notions in Algoritmics

Outline

Basic Notions to Design and Analyze Algorithms

- Notation and terminology
- Machine models
- Pseudo-code
- Analysis of algorithms
- Computational complexity

Good Algorithms

We say that an algorithm A is

Efficient = good = polynomial time = polytime
iff
there exists $p(n)$ such that $T(A) = O(p(n))$

There are problems for which no polytime algorithm is known. This course is about those problems.

Complexity theory classifies problems

Computational Complexity

Equivalent Notions
Consider Decision Problems

- A problem Π is in $\mathcal{P}$ if $\exists$ algorithm A that finds a solution in polynomial time.
- in $\mathcal{NP}$ if $\exists$ verification algorithm $A(s, k)$ that verifies a binary certificate (whether it is a solution to the problem) in polynomial time.
- Polynomial time reduction formally shows that one problem Π_1 is at least as hard as another Π_2, to within a polynomial factor. (there exists a polynomial time transformation) $\Pi_2 \leq_P \Pi_1 \Rightarrow \Pi_2$ is no more than a polynomial harder than Π_1.
- Π_1 is in $\mathcal{NP}$-complete if
  1. $\Pi_1 \in \mathcal{NP}$
  2. $\forall \Pi_2 \in \mathcal{NP}\ \Pi_2 \leq_P \Pi_1$
- If Π_1 satisfies property 2, but not necessarily property 1, we say that it is $\mathcal{NP}$-hard:

Important concepts (continued):

- $\mathcal{NP}$: Class of problems that can be solved in polynomial time by a non-deterministic machine.

  Note: non-deterministic $\neq$ randomized; non-deterministic machines are idealized models of computation that have
  the ability to make perfect guesses.

- $\mathcal{NP}$-complete: Among the most difficult problems in $\mathcal{NP}$; believed to have at least exponential time-complexity for any realistic machine or programming model.

- $\mathcal{NP}$-hard: At least as difficult as the most difficult problems in $\mathcal{NP}$, but possibly not in $\mathcal{NP}$ (*i.e.*, may have even worse complexity than $\mathcal{NP}$-complete problems).

Many combinatorial problems are hard but some problems can be solved efficiently

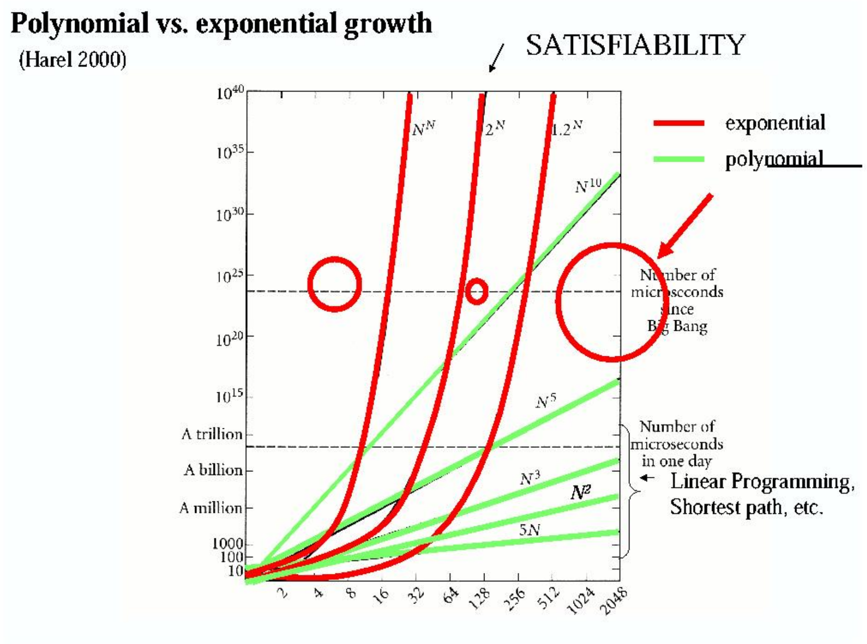
- Longest path problem is $\mathcal{NP}$-hard
  but not shortest path problem
- SAT for 3-CNF is $\mathcal{NP}$-complete
  but not 2-CNF (linear time algorithm)
- TSP is $\mathcal{NP}$-hard, the associated decision problem (for any solution quality) is $\mathcal{NP}$-complete
  but not the Euler tour problem
- TSP on Euclidean instances is $\mathcal{NP}$-hard
  but not where all vertices lie on a circle.

An online compendium on the computational complexity of optimization problems:
`http://www.nada.kth.se/~viggo/problemlist/compendium.html`

Application Scenarios

Practically solving hard combinatorial problems:

- Average-case *vs* worst-case complexity
  (*e.g.* Simplex Algorithm for linear optimization);
- Approximation of optimal solutions:
  sometimes possible in polynomial time (*e.g.*, Euclidean TSP),
  but in many cases also intractable (*e.g.*, general TSP);
- Randomized computation is often practically
  (and possibly theoretically) more efficient;
- Asymptotic bounds *vs* true complexity:
  constants matter!



Approximation Algorithms

Definition: Approximation Algorithms

An algorithm $\mathcal{A}$ is said to be a δ-approximation algorithm if it runs in *polynomial* time and for every problem instance π with optimal solution value $\mathrm{OPT}(\pi)$

$$\text{minimization:} \quad \frac{\mathcal{A}(\pi)}{\mathrm{OPT}(\pi)} \leq \delta \quad \delta \geq 1$$

$$\text{maximization:} \quad \frac{\mathcal{A}(\pi)}{\mathrm{OPT}(\pi)} \geq \delta \quad \delta \leq 1$$

(δ is called *worst case bound*, *worst case performance*, *approximation factor*, *approximation ratio*, *performance bound*, *performance ratio*, *error ratio*)

Definition: Polynomial approximation scheme

A family of approximation algorithms for a problem Π, $\{\mathcal{A}_\epsilon\}_\epsilon$, is called a polynomial approximation scheme (PAS), if algorithm $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation algorithm and its running time is polynomial in the size of the input for a fixed ϵ

Definition: Fully polynomial approximation scheme

A family of approximation algorithms for a problem Π, $\{\mathcal{A}_\epsilon\}_\epsilon$, is called a fully polynomial approximation scheme (FPAS), if algorithm $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation algorithm and its running time is polynomial in the size of the input and $1/\epsilon$

Randomized Algorithms

Definition: Randomized Algorithms

Their running time depends on the random choices made.
Hence, the running time is a random variable.

In the case of randomized optimization heuristics
solution quality is also a random variable.

We distinguish:

- single-pass heuristics (denoted $\mathcal{A}^\perp$): have an embedded termination, for example, upon reaching a certain state

  (generalized optimization Las Vegas algorithms [B2])

- asymptotic heuristics (denoted $\mathcal{A}^\infty$): do not have an embedded termination and they might improve their solution asymptotically

  (both probabilistically approximately complete
  and essentially incomplete [B2])

Outline

Graphs

Graphs are combinatorial structures useful to model several applications

Terminology:

- $G = (V, E)$, $E \subseteq V \times V$, vertices, edges, $n = |V|$, $m = |E|$, digraphs, undirected graphs, subgraph, induced subgraph
- $e = (u, v) \in E$, e incident on u and v; u, v adjacent, edge weight or cost
- particular cases often omitted: self-loops, multiple parallel edges
- degree, δ, Δ, outdegree, indegree
- path $P =< v_0, v_1, \ldots, v_k >$, $(v_0, v_1) \in E, \ldots, (v_{k-1}, v_k) \in E$, $< v_0, \ldots, v_1 >$ has length 2, $< v_0, v_1, v_2, v_0 >$ cycle,
- directed acyclic digraph
- digraph strongly connected ($\forall u, v \; \exists (uv)$-path), strongly connected components
- G is a tree ($\exists$ path between any two vertices) $\iff$ G is connected and has $n - 1$ edges $\iff$ G is connected and contains no cycles.
- parent, children, sibling, height, depth

Representing Graphs

Operations:

- access associated information
- Navigation: access outgoing edges
- Edge queries: given u and v is there an edge?
- Update: add remove edges, vertices

Data Structures:

- Edge sequences
- Adjacency arrays
- Adjacency lists
- Adjacency matrix

How to choose?

- it depends on the graphs and the application
- if time and space not crucial no need to customize the structures
- use interfaces that make easy to change the data structure
- libraries offer different choices (LEDA, Java `jdsl.graph`)

Useful Graph Algorithms

- Strongly connected components
- Matching
- Shortest Path
- Minimum Spanning Tree
- Max flow - Min cut

Outline

Methods and Algorithms

A Method is a general framework for the development of a solution algorithm. It is not problem-specific.

An Algorithmic model (or simply algorithm) is the instantiation of a method on a certain problem Π.
The level of instantiation may vary:

- minimally instantiated (few details, algorithm template)
- lowly instantiated (which data structure to use)
- highly instantiated (programming tricks that give speedups)
- maximally instantiated (details specific of a programming language and computer architecture)

A Program is the formulation of an algorithm in a programming language.

Solution Methods

- **Exact methods**:
  complete: guaranteed to eventually find (optimal) solution,
  or to determine that no solution exists (eg, systematic enumeration)
  - Search algorithms (backtracking, branch and bound)
  - Dynamic programming
  - Constraint programming
  - Integer programming
  - Dedicated Algorithms
- **Approximation methods**
  worst-case solution guarantee
  http://www.nada.kth.se/~viggo/problemlist/compendium.html
- **Heuristic (Approximate) methods**:
  incomplete: not guaranteed to find (optimal) solution,
  and unable to prove that no solution exists
  - Integer programming relaxations
  - Randomized backtracking
  - Heuristic algorithms

Outline

Generic methods:

- Do not achieve same performance as specific algorithms
- Allow to save development time