Lecture 1

Course Introduction
Combinatorial Optimization and Problem Solving

Marco Chiarandini

Deptartment of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

Course Presentation

- Schedule (28 lecture hours):
  - Tuesday 8.15-10
  - Wednesday 10-11.45 or 10.15-12 ?
  - Last lecture: Wednesday, 9th October, 2008
- Communication tools
  - Course Public Web Site (Ws) $\Leftrightarrow$ Blackboard (Bb)
  - Announcements (Bb)
    (link from `http://www.imada.sdu.dk/~marco/DM811/`)
  - Discussion board (Bb)
  - Personal email (Bb)
  - You are welcome to visit me in my office in working hours (8-16).
- Course content

Course Presentation

- Evaluation: final individual project (internal examiner)
  - Algorithm design
  - Implementation (deliverable and checkable source code)
  - (Analytical) and experimental analysis
  - Written description
  - Performance counts

Course Material

- Books:

  B1 *Theoretical Aspects of Local Search*. W. Michiels, E. Aarts and J. Korst. Springer Berlin Heidelberg, 2007

- Photocopies (from Course Documents left menu of Blackboard)

  B2 *Artificial Intelligence: A Modern Approach*. S. Russell and P. Norvig. Prentice Hall, 2003

  B3 *Stochastic Local Search: Foundations and Applications*, H. Hoos and T. Stützle, 2005, Morgan Kaufmann

  B4 *Search methodologies: introductory tutorials in optimization and decision support techniques* E.K. Burke, G. Kendall, 2005, Springer, New York

- Articles from the web site
- R notes from the web site
- Lecture slides
- Assignments

- ...but take notes in class!

Course Presentation

Practical experience is important to learn to use heuristics
Implementation details play an important role.

- Running Assignment
  - Home preparation
  - Implementation of heuristics for a certain problem
  - Experimental analysis of performance
  - Groups in competition
  - (worthwhile in preparation of the project!)
- Further Assignment Sheets
- Problem solving in class

Personal Lecture Journal

aka Personal Blog

Content:

- Write the main points of the lecture in an appropriate language
- Publish your results on the common problem
- Other topics indicated in Announcements

Functions:

- Revise the lecture
- Let me have a feedback whether the goals of the lecture have been achieved.

Outline

Combinatorial Problems

Combinatorial problems

They arise in many areas
of Computer Science, Artificial Intelligence
and Operations Research:

- allocating register memory
- planning, scheduling, timetabling
- Internet data packet routing
- protein structure prediction
- combinatorial auctions winner determination
- portfolio selection
- ...

Combinatorial Problems (2)

Simplified models are often used to formalize real life problems

- finding shortest/cheapest round trips (TSP)
- finding models of propositional formulae (SAT)
- coloring graphs (GCP)
- finding variable assignment which satisfy constraints (CSP)
- partitioning graphs or digraphs
- partitioning, packing, covering sets
- finding the order of arcs with minimal backward cost
- ...

Example Problems

- They are chosen because conceptually concise, intended to illustrate the development, analysis and presentation of algorithms
- Although real-world problems tend to have much more complex formulations, these problems capture their essence

Combinatorial Problems (3)

Combinatorial problems are characterized by an input,
i.e., a general description of conditions and parameters and
a question (or task, or objective) defining
the properties of a solution.

They involve finding a grouping, ordering, or assignment
of a discrete, finite set of objects that satisfies given conditions.

(Candidate) solutions are combinations of objects or solution components
that need not satisfy all given conditions.

Solutions are candidate solutions that satisfy all given conditions.

Combinatorial Problems (4)

Classical Example

Traveling Salesman Problem

- **Given:** edge-weighted, undirected graph G
- **Task:** find a minimal-weight Hamiltonian cycle in G.

Note:

- **solution component:** segment consisting of two points that are visited one directly after the other
- **candidate solution:** one of the $(n-1)!$ possible sequences of points to visit one directly after the other.
- **solution:** Hamiltonian cycle of minimal length

Decision problems

Hamiltonian cycle problem

- **Given:** undirected graph G
- **Question:** does G contain a Hamiltonian cycle?

solutions = candidate solutions that satisfy given *logical conditions*

Two variants:

- Existence variant: Determine whether solutions for given problem instance exists
- Search variant: Find a solution for given problem instance (or determine that no solution exists)

Optimization problems

Traveling Salesman Problem

- **Given:** edge-weighted, undirected graph G
- **Task:** find a minimal-weight Hamiltonian cycle in G.

- objective function measures solution quality (often defined on all candidate solutions)
- find solution with optimal quality, *i.e.*, minimize/maximize obj. func.

Variants of optimization problems:

- Search variant: Find a solution with optimal objective function value for given problem instance
- Evaluation variant: Determine optimal objective function value for given problem instance

Remarks

- Every optimization problem has associated decision problems:
  Given a problem instance and a fixed solution quality bound b,
  find a solution with objective function value $\leq b$ (for minimization problems) or determine that no such solution exists.
- Many optimization problems have an objective function as well as constraints (= logical conditions) that solutions must satisfy.
- A candidate solution is called feasible (or valid) iff it satisfies the given constraints.
- Approximate solutions are feasible candidate solutions that are not optimal. (to be refined later).
- **Note:** Logical conditions can always be captured by an objective function such that feasible candidate solutions correspond to solutions of an associated decision problem with a specific bound.

Combinatorial Problems (5)

General problem *vs* problem instance:

General problem Π:

- Given *any* set of points X, find a Hamiltonian cycle
- *Solution:* Algorithm that finds shortest Hamiltonian cycle for any X

Problem instantiation $\pi = \Pi(I)$:

- Given a specific set of points I, find a shortest Hamiltonian cycle
- *Solution:* Shortest Hamiltonian cycle for I

Problems can be formalized on sets of problem instances $\mathcal{I}$

Traveling Salesman Problem

Types of TSP instances:

- Symmetric: For all edges uv of the given graph G, vu is also in G, and $w(uv) = w(vu)$.
  Otherwise: asymmetric.
- Euclidean: Vertices = points in an Euclidean space,
  weight function = Euclidean distance metric.
- Geographic: Vertices = points on a sphere,
  weight function = geographic (great circle) distance.

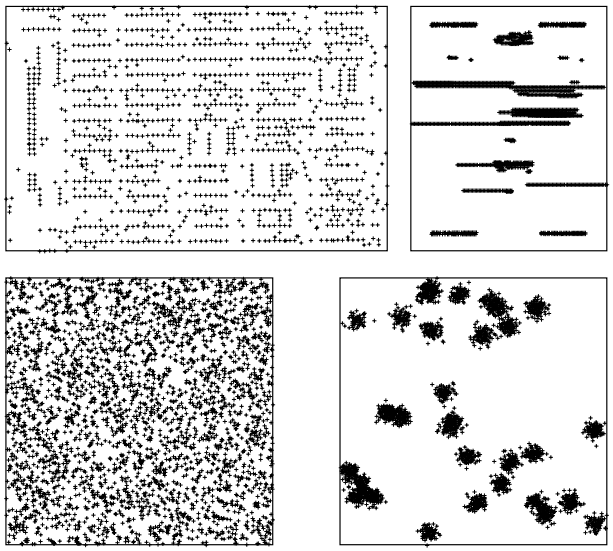
TSP: Benchmark Instances

Instance classes

- Real-life applications (geographic, VLSI)
- Random Euclidean
- Random Clustered Euclidean
- Random Distance

Available at the TSPLIB (more than 100 instances upto 85.900 cities)
and at the 8th DIMACS challenge

TSP: Instance Examples



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

Methods and Algorithms

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

An Algorithm (or algorithmic model) is a problem-specific template that leaves some practical details unspecified.
The level of detail 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.

An algorithm can thus be regarded as a class of computer programs (its implementations)

Problem specific methods:

- Dynamic programming (knapsack)
- Dedicated algorithms (shortest path)

Generic methods:

- Integer Programming (knapsack)
- Constraint Programming (constraint satisfaction problem)

Generic methods:

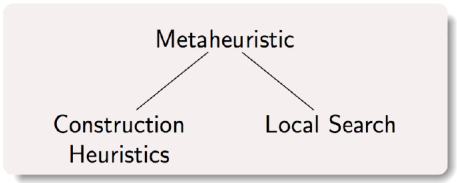
👍 Allow to save development time

👎 Do not achieve same performance as specific algorithms

Heuristics

Get inspired by approach to problem solving in human mind (more on this later) [Newell and Simon, 1976]

- effective rules
- trial and error



Applications:

- Optimization, Timetabling, Routing, Scheduling
- But also in Psychology, Economics, Management

Side aspects: basis on empirical evidence rather than mathematical logic. Getting things done in the given time. Good having creativity in problem solving and criticism.

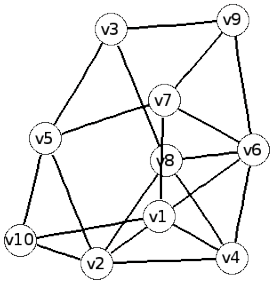
Outline

The Vertex Coloring Problem

Given: A graph G and a set of colors Γ.

A proper coloring is an assignment of one color to each vertex of the graph such that adjacent vertices receive different colors.

Decision version (k-coloring)
Task: Find a proper coloring of G that uses at most k colors.
Optimization version (chromatic number)
Task: Find a proper coloring of G that uses the minimal number of colors.



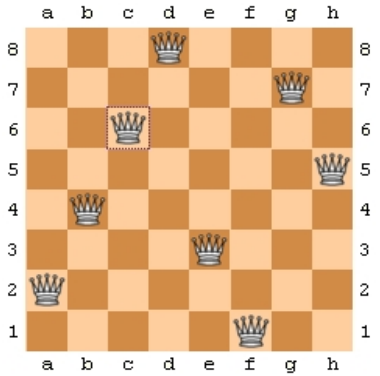
Design an algorithm for solving general instances of the graph coloring problem.

Home Assignment

N-Queens problem

Input: A chessboard of size N × N

Task: Find a placement of n queens on the board such that no two queens are on the same row, column, or diagonal.



Home Assignment

N^2 Queens – queen graph coloring problem

Input: A chessboard of size N × N

Question: Given such a chessboard, is it possible to place N sets of N queens on the board so that no two queens of the same set are in the same row, column, or diagonal?

0	5	9	6	3	8	4	1	10	11	7	2
7	11	4	2	1	6	10	3	0	8	9	5
8	1	10	9	5	2	0	7	11	6	3	4
10	0	3	8	7	11	9	5	4	1	2	6
5	6	11	4	2	1	3	0	8	9	10	7
11	7	0	1	10	4	8	6	3	2	5	9
2	8	6	3	9	5	7	11	1	10	4	0
3	4	5	0	11	10	6	9	2	7	8	1
9	2	1	10	4	7	5	8	6	3	0	11
4	10	7	11	0	3	1	2	9	5	6	8
6	3	2	5	8	9	11	4	7	0	1	10
1	9	8	7	6	0	2	10	5	4	11	3

The answer is yes $\iff$ the graph admits a coloring with N colors.

Problem Solving

Problem solving is a mental process considered the most complex of all intellectual functions.

Move from a given state to a desired goal state using the knowledge we have. However often solutions seem to be original and creative.

Theories:

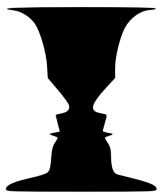
1. Gestalt approach
2. Problem space theory (Information-processing theory)

Gestalt Approach

The process of problem solving is

Behaviourists: reproduction of known responses, trial and error process

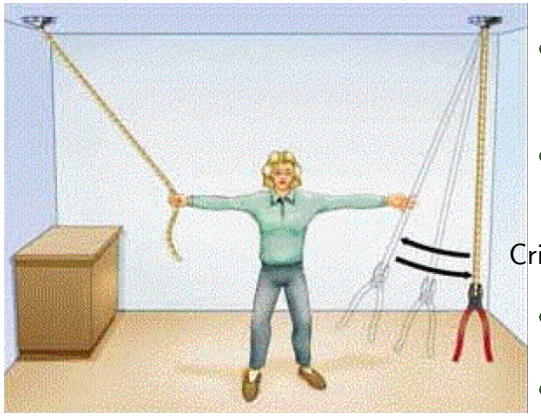
Gestalt school: (German psychologists in 20-30's concerned with experience as a whole rather than composed of parts)



- reproductive: draws on previous experience may cause fixation and hinder the solution
- productive: insight and problem restructuring

Gestalt Approach

Maier's Experiment (1931): pendulum problem



- Those who solved it rarely reported the cue
- Unconscious clue can lead to problem restructuring and insight

Criticism:

- unspecified and vague
- descriptive nature, not normative or explanatory (what processes are involved?)

Representational Theory

Incorporate Gestalt ideas into a working theory [Ohlsson, 1992]

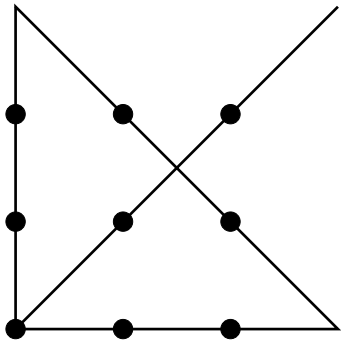
- A problem is represented in a certain way in the person's mind and this serves as a source of information from long-term memory
- The retrieval process spreads activation over relevant long term memory items
- A block occurs if the way a problem is represented does not lead to a helpful memory search
- The way the problem is represented changes and the memory search is extended, making new information available
- Representational change can occur due to elaboration (addition of new information) constraint relaxation (rules are reinterpreted) or re-encoding (fixedness is removed)
- Insight occurs when a block is broken and retrieved knowledge results in solution

Representational Theory

Draw four straight lines to join all the dots without taking the pen off the page

This problem was given to employees at Disney as is reportedly the origin of the expression "thinking outside the box"

Who failed probably did not consider extending the lines beyond the grid
➡ Constraint relaxation



Problem Space Approach

Information-processing theory:

> [A. Newell and H.A. Simon. *Computer science as empirical inquiry: symbols and search*. Communications of the ACM, 1976]

- generating problems states in the problem space using legal transition operators to go from an initial state to a goal state.
- restrictions imposed by human processing system (limited short-term memory and speed).
- maximization heuristic: reduce difference between initial and goal state.
- progress monitoring: assessment of rate of progress

Further Elements

- Experience helps us since we can learn how to structure problems space and appropriate operators.
- Analogy (old knowledge is used to solve new problems)
- Domain knowledge and skill acquisition
  - Observation of expert vs novice in chess
    - chess masters remember board configurations
    - structure available to maintain configurations in short term memory
    - grouping of problems according to underlying conceptual similarities
    - better encoding of knowledge and easier information retrieval
  - skill acquisition:
    - general-purpose rules
    - rules to specific task
    - rules are tuned to speed up

Further reading:

- A. Newell and H.A. Simon. Computer science as empirical inquiry: symbols and search. Communications of the ACM, ACM, 1976, 19(3), 113-126
- A. Dix, J. Finlay, G.D. Abowd and R. Beale. Human-Computer Interaction. Pearson, Prentice Hall, 2004. (Chapter 1)
- Ormerod, T. MacGregor, J. Chronicle, E. (2002) Dynamics and Constraints in Insight Problem Solving. Journal of Experimental Psychology: Learning, Memory, and Cognition vol. 28 (4) pp 791-799

The Mathematical Perspective

Beside psychologists, also mathematicians reflected upon problem solving processes:

- George Pólya, *How to Solve it*, 1945
- J. Hadamard, *The Mathematician's Mind - The Psychology of Invention in the Mathematical Field*, 1945

Mathematical Problem Solving
George Pólya

George Pólya's 1945 book *How to Solve It*:

1. Understand the problem.
2. Make a plan.
3. Carry out the plan.
4. Look back on your work. How could it be better?

`http://en.wikipedia.org/wiki/How_to_Solve_It`

Pólya's First Principle: Understand the Problem

- Do you understand all the words used in stating the problem?
- What are you asked to find or show?
- Is there enough information to enable you to find a solution?
- Can you restate the problem in your own words?
- Can you think of a picture or a diagram that might help you to understand the problem?

Pólya's Second Principle: Devise a plan

There are many reasonable ways to solve problems.

- Guess and check
- Make an orderly list
- Eliminate possibilities
- Use symmetry
- Consider special cases
- Use direct reasoning

Also suggested:

- Look for a pattern
- Draw a picture
- Solve a simpler problem
- Use a model
- Work backward

Choosing an appropriate strategy is best learned by solving many problems.

Pólya's Third Principle: Carry out the plan

"Needed is care and patience, given that you have the necessary skills. Persist with the plan that you have chosen. If it continues not to work discard it and choose another. Don't be misled, this is how mathematics is done, even by professionals."

Pólya's Fourth Principle: Review/Extend

"Much can be gained by taking the time to reflect and look back at what you have done, what worked and what didn't. Doing this will enable you to predict what strategy to use to solve future problems."

Heuristic	Informal Description	Formal analogue
Analogy	Can you find a problem analogous to your problem and solve that?	Map
Generalization	Can you find a problem more general than your problem?	Generalization
Induction	Can you solve your problem by deriving a generalization from some examples?	Induction
Variation of the Problem	Can you vary or change your problem to create a new problem (or set of problems) whose solution(s) will help you solve your original problem?	Search
Auxiliary Problem	Can you find a subproblem or side problem whose solution will help you solve your problem?	Subgoal
Here is a problem related to yours and solved before	Can you find a problem related to yours that has already been solved and use that to solve your problem?	Pattern recognition Pattern matching Reduction
Specialization	Can you find a problem more specialized?	Specialization
Decomposing and Recombining	Can you decompose the problem and "recombine its elements in some new manner"?	Divide and conquer
Working backward	Can you start with the goal and work backwards to something you already know?	Backward chaining
Draw a Figure	Can you draw a picture of the problem?	Diagrammatic Reasoning [3]
Auxiliary Elements	Can you add some new element to your problem to get closer to	Extension

Inspiration can strike anytime, particularly after an individual had worked hard on a problem for days and then turned the attention to another activity.

The Mathematician's Mind - The Psychology of Invention in the Mathematical Field, J. Hadamard, 1945

Outline

Concepts from Algorithmics

Outline

- Notation of runtime analysis
- Machine models
- Pseudo-code
- Computational complexity
- Analysis of algorithms
- Graphs

Motivations

Questions:

1. How hard, computationally, is a given a problem to solve using the most efficient algorithm for that problem?
2. How good is the algorithm designed?

1. Complexity theory
2. Asymptotic notation, running time bounds
   Approximation theory

Asymptotic notation

$n \in \mathbf{N}$ instance size

max time	worst case	$T(n) = \max\{T(\pi) : \pi \in \Pi_n\}$
average time	average case	$T(n) = \frac{1}{\lvert\Pi_n\rvert}\{\sum_\pi T(\pi) : \pi \in \Pi_n\}$
min time	best case	$T(n) = \min\{T(\pi) : \pi \in \Pi_n\}$

Growth rate or asymptotic analysis

$f(n)$ and $g(n)$ same growth rate if	$c \leq \frac{f(n)}{g(n)} \leq d$ for n large
$f(n)$ grows faster than $g(n)$ if	$f(n) \geq c \cdot g(n)$ for all c and n large

big O	$O(f) = \{g(n) : \exists c > 0, \forall n > n_0 : g(n) \leq c \cdot f(n)\}$
big omega	$\Omega(f) = \{g(n) : \exists c > 0, \forall n > n_0 : g(n) \geq c \cdot f(n)\}$
theta	$\Theta(f) = O(f) \cap \Omega(f)$
little O	$o(f) = g$ grows strictly more slowly

Machine model

For asymptotic analysis we use RAM machine

- single processor unit
- all memory access take same amount of time

It is an abstraction from machine architecture: it ignores caches, memories hierarchies, parallel processing (SIMD, multi-threading), etc.

Total execution of a program = total number of instructions executed

We are not interested in constant and lower order terms

Pseudo-code

We express algorithms in natural language and mathematical notation, and in pseudo-code, which is an abstraction from programming languages C, C++, Java, etc.

(In implementation you can choose your favorite language)

Programs must be correct.
Certifying algorithm: computes a certificate for a post condition (without increasing asymptotic running time)

Good Algorithms

We say that an algorithm A is

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

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

Complexity theory classifies problems

Computational Complexity

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:

Summary

1. Course Introduction
2. Combinatorial Optimization
   - Combinatorial Problems, Terminology
   - Solution Methods, Overview
   - Travelling Salesman Problem
3. Problem Solving
   - Example: Graph Coloring Problem
   - Psychological Perspective
   - Mathematical Perspective, Polya's view
4. Basic Concepts from Algorithmics

Outlook

Next Time:

- Generalitis on Heuristics
- Working Environment
- Basic Concepts from Algorithmics

In preparation:

- Check Announcements
- Set up and Write in the Blog
- Think individually to the Running Assignment and be prepared to discuss your solutions
- Read the chapters in photocopies