DM811 – Autumn 2011

Heuristics for Combinatorial Optimization

Lecture 1

Course Introduction
Combinatorial Optimization and Modeling

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

Schedule and Material

- Schedule (28 lecture hours):
  - Monday 12:15-14:00
  - Tuesday 10:15-12:00
  - Thursday 12.15-14:00
  - Last lecture: Thursday, October 13, 2011

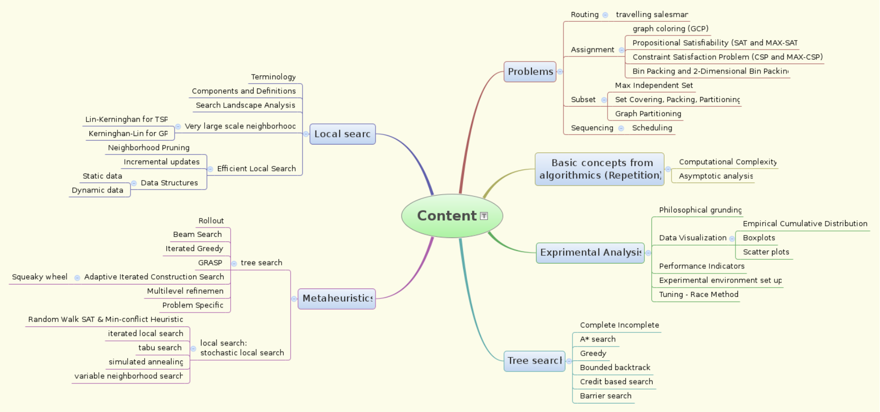
Schedule and Material

- Schedule (28 lecture hours):
  - Monday 12:15-14:00
  - Tuesday 10:15-12:00
  - Thursday 12.15-14:00
  - Last lecture: Thursday, October 13, 2011

- Communication tools
  - Course Public Webpage (Wp) ⇔ BlackBoard (BB)
    (link from http://www.imada.sdu.dk/~marco/DM811/)
  - Announcements in BlackBoard
  - Course Documents (Photocopies) in (BB)
  - Discussion Board in (BB)
  - Assignment Hand in in (BB)
  - Personal email

Contents

Heuristic algorithms: compute, efficiently, good solutions to a problem with no guarantee of optimality.



Evaluation

- Obligatory Assignments, pass/fail, evaluation by teacher ($\approx$ 4 hand ins)
- Evaluation: final individual project, 7-grade scale, external examiner)
  - Algorithm design
  - Implementation (deliverable and checkable source code)
  - (Analytical) and experimental analysis
  - Written description
  - Performance counts

References

- Main References:
  - B1 W. Michiels, E. Aarts and J. Korst. Theoretical Aspects of Local Search. Springer Berlin Heidelberg, 2007
  - B2 S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. (Part II) Prentice Hall, 2003.
  - B3 Comet Tutorial (see doc in Comet Application)
  - B4 P.V. Hentenryck and L. Michel. Constraint-Based Local Search. The MIT Press, Cambridge, USA, 2005.
  - B5 H. Hoos and T. Stuetzle, Stochastic Local Search: Foundations and Applications, 2005, Morgan Kaufmann
  - B6 E.K. Burke, G. Kendal, Search methodologies: introductory tutorials in optimization and decision support techniques 2005, Springer, New York
- Photocopies (from Course Documents left menu of BlackBoard)
- Articles from the Webpage
- R notes from the Webpage
- Lecture slides
- Assignments and Exercises
- ...but take notes in class!

Active participation

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

- Be prepared for:
  - Problem solving in class
  - Assignment Sheets for hands on experience $\leadsto$ programming
  - Experimental analysis of performance
  - Group discussions
  - Exercise Sheets

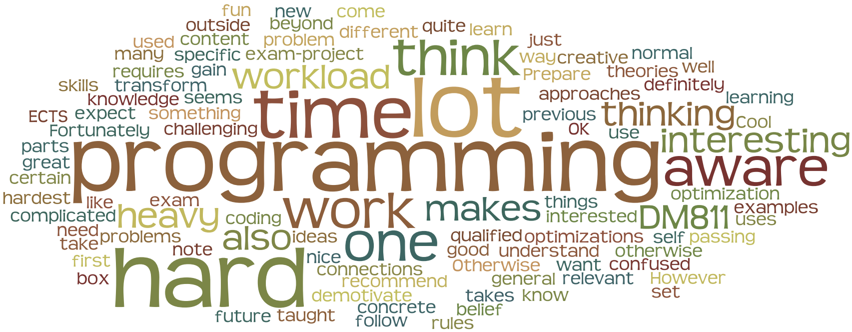
Require home preparation!
Worthwhile in preparation of the project!

Former students' feedback (1/2)

On the course:

- the course bulids on a lot of knowledge from previous courses
- programming
- practical drive
- taught on examples
- no sharp rules are given and hence more space left to creativity
- unexpected heavy workload

Word cloud


programming
hard
time
lot
think
work
one
aware
interesting
thinking
workload
makes
heavy
also
DM811
fun
new
come
outside
beyond
different
quite
learn
used
content
problem
just
many
specific
exam-project
way
creative
normal
requires
gain
Prepare
theories
well
skills
transform
definitely
knowledge
seems
approaches
learning
ECTS
expect
something
previous
cool
Fortunately
challenging
OK
use
parts
great
certain
optimization
hardest
like
exam
things
examples
complicated
coding
interested
uses
need
problems
ideas
qualified
optimizations
self
passing
take
note
good
understand
otherwise
first
nice
Otherwise
want
confused
box
connections
recommend
general
relevant
However
demotivate
takes
know
set
concrete
belief
future
taught
follow
rules

Former students' feedback (2/2)

On the exam:

- hardest part is the design of the heuristics
  the content of the course is vast $\rightsquigarrow$ many possibilities without clue on what will work best.

In general:

- Hands-on examples are relevant, would be nice closer look at source code.

From my side, mistakes I would like to see avoided:

- non competitive local search procedures and mistaken data aggregation in instance set analysis.

Outline

Outline

Combinatorial Problems (1/5)

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 (2/5)

Simplified models are often used to formalize real life problems

- finding models of propositional formulae (SAT)
- finding variable assignment which satisfy constraints (CSP)
- coloring graphs (GCP)
- finding shortest/cheapest round trips (TSP)
- 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 (3/5)

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

Combinatorial Problems (3/5)

Combinatorial problems are characterized by an input,
i.e., a general description of conditions (or constraints) 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.

Combinatorial Problems (3/5)

Combinatorial problems are characterized by an input,
i.e., a general description of conditions (or constraints) 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.

Warning, in this course,

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

Combinatorial Problems (3/5)

Combinatorial problems are characterized by an input,
i.e., a general description of conditions (or constraints) 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.

Warning, in this course,

(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 (4/5)

Classical Example

Traveling Salesman Problem

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

Combinatorial Problems (4/5)

Classical Example

Traveling Salesman Problem

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

Note:

- **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?

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 complete graph G
- **Task:** find a minimum-weight Hamiltonian cycle in G.

Optimization problems

Traveling Salesman Problem

- **Given:** edge-weighted, undirected complete graph G
- **Task:** find a minimum-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.

Optimization problems

Traveling Salesman Problem

- **Given:** edge-weighted, undirected complete graph G
- **Task:** find a minimum-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:

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

Remarks

- Every optimization problem has an associated decision problem: 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.

Remarks

- Every optimization problem has an associated decision problem:
  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.

Remarks

- Every optimization problem has an associated decision problem:
  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.

Remarks

- Every optimization problem has an associated decision problem: 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.

Remarks

- Every optimization problem has an associated decision problem: 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.
- **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 (5/5)

General problem *vs* problem instance:

General problem Π:

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

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

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

Problems can be formalized on sets of problem instances $\mathcal{I}$ (instance classes)

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.

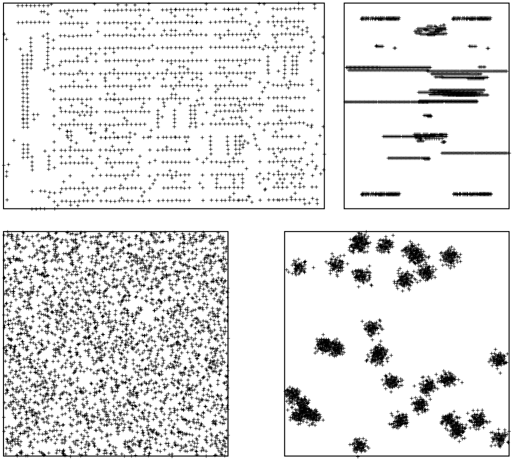
TSP: Benchmark Instances

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



Outline

Methods and Algorithms

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

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 only some practical details unspecified.

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 only 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)

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 only 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)

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

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

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

Problem specific methods:

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

General methods:

- Integer Programming
- Constraint Programming

Problem specific methods:

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

General methods:

- Integer Programming
- Constraint Programming

Generic methods:

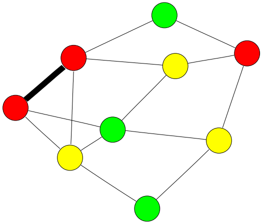
- Allow to save development time
- Do not achieve same performance as specific algorithms

Outline

The Vertex Coloring Problem

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

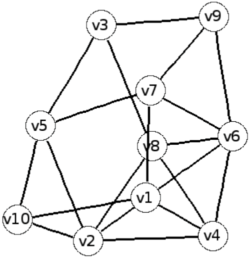
A proper coloring is an assignment of one color to each vertex of the graph such that adjacent vertices receive different colors.



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.

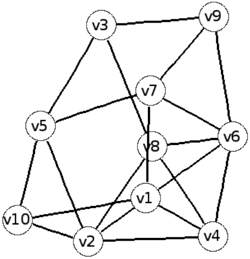


The Vertex Coloring Problem

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.

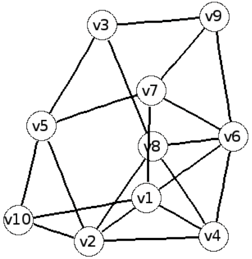


The Vertex Coloring Problem

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.



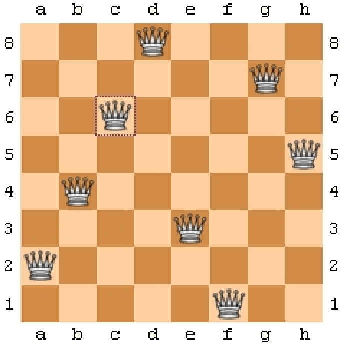
Design an algorithm for solving general instances of the graph coloring problem.

Home Assignment

N-Queens problem

Input: A chessboard of size $N \times N$

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

Input: A chessboard of size $N \times N$

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

Home Assignment

N^2 Queens

Input: A chessboard of size $N \times N$

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?

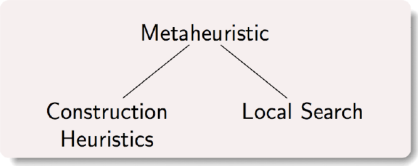
0	5	9	6	3	8	4	1	10	11	7	2
7	11	4	2	1	6	10	3	0	8	9	5
8	1	10	9	5	2	0	7	11	6	3	4
10	0	3	8	7	11	9	5	4	1	2	6
5	6	11	4	2	1	3	0	8	9	10	7
11	7	0	1	10	4	8	6	3	2	5	9
2	8	6	3	9	5	7	11	1	10	4	0
3	4	5	0	11	10	6	9	2	7	8	1
9	2	1	10	4	7	5	8	6	3	0	11
4	10	7	11	0	3	1	2	9	5	6	8
6	3	2	5	8	9	11	4	7	0	1	10
1	9	8	7	6	0	2	10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

The answer is yes $\iff$ an opportune conflict graph admits a coloring with N colors

Heuristics

Get inspired by approach to problem solving in human mind
[Newell and Simon, 1976]

- effective rules
- trial and error



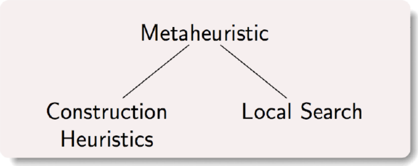
Applications:

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

Heuristics

Get inspired by approach to problem solving in human mind
[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 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

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 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 a solution?	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

solution algorithm = model + search

Outline

IP-models

Standard IP formulation: Let x_{vk} be a 0–1 variable equal to 1 whenever the vertex v takes the color k
and y_k be 1 if color k is used and 0 otherwise

$$
\begin{aligned}
\min \quad & \sum_{k \in K} y_k \\
\text{s.t.} \quad & \sum_{k \in K} x_{vk} = 1, && \forall v \in V, \\
& x_{vk} + x_{uk} \leq y_k, && \forall (u,v) \in E(G), \forall k \in K, \\
& x_{vk} \in \{0,1\}, && \forall v \in V, \forall k \in K \\
& y_k \in \{0,1\}, && \forall k \in K.
\end{aligned}
$$

Column generation formulation

- Notation
  - Independent set s, with cardinality c_s
  - S: Collection of every maximal independent set of G
  - S_v: subset of S that contains v
  - λ_s: 0-1 variable equal to 1 if independent set s is used

Column generation formulation

- Notation
  - Independent set s, with cardinality c_s
  - S: Collection of every maximal independent set of G
  - S_v: subset of S that contains v
  - λ_s: 0-1 variable equal to 1 if independent set s is used

$$
\begin{aligned}
\min \quad & \sum_{s \in \mathcal{S}} \lambda_s \\
\text{s.t.} \quad & \sum_{s \in \mathcal{S}_v} \lambda_s \geq 1, && \forall v \in V, \\
& \lambda_s \in \{0, 1\}, && \forall s \in \mathcal{S}.
\end{aligned}
$$

Outline

Constraint Programming

The **domain** of a variable x, denoted $D(x)$, is a finite set of elements that can be assigned to x.

Constraint Programming

The **domain** of a variable x, denoted $D(x)$, is a finite set of elements that can be assigned to x.

A **constraint** C on X is a subset of the Cartesian product of the domains of the variables in X, i.e., $C \subseteq D(x_1) \times \cdots \times D(x_k)$. A tuple $(d_1, \ldots, d_k) \in C$ is called a solution to C.

Constraint Programming

The **domain** of a variable x, denoted $D(x)$, is a finite set of elements that can be assigned to x.

A **constraint** C on X is a subset of the Cartesian product of the domains of the variables in X, i.e., $C \subseteq D(x_1) \times \cdots \times D(x_k)$. A tuple $(d_1, \ldots, d_k) \in C$ is called a solution to C.
Equivalently, we say that a solution $(d_1, ..., d_k) \in C$ is an assignment of the value d_i to the variable $x_i, \forall 1 \leq i \leq k$, and that this assignment satisfies C.

Constraint Programming

The **domain** of a variable x, denoted $D(x)$, is a finite set of elements that can be assigned to x.

A **constraint** C on X is a subset of the Cartesian product of the domains of the variables in X, i.e., $C \subseteq D(x_1) \times \cdots \times D(x_k)$. A tuple $(d_1, \ldots, d_k) \in C$ is called a solution to C.
Equivalently, we say that a solution $(d_1, ..., d_k) \in C$ is an assignment of the value d_i to the variable $x_i, \forall 1 \leq i \leq k$, and that this assignment satisfies C.
If $C = \emptyset$, we say that it is inconsistent.

Constraint Programming

Constraint Satisfaction Problem (CSP)

A CSP is a finite set of variables X, together with a finite set of constraints C, each on a subset of X. A **solution** to a CSP is an assignment of a value $d \in D(x)$ to each $x \in X$, such that all constraints are satisfied simultaneously.

Constraint Programming

Constraint Satisfaction Problem (CSP)

A CSP is a finite set of variables X, together with a finite set of constraints C, each on a subset of X. A **solution** to a CSP is an assignment of a value $d \in D(x)$ to each $x \in X$, such that all constraints are satisfied simultaneously.

Constraint Optimization Problem (COP)

A COP is a CSP P defined on the variables $x_1, \ldots, x_n$, together with an objective function $f : D(x_1) \times \cdots \times D(x_n) \to Q$ that assigns a value to each assignment of values to the variables. An **optimal solution** to a minimization (maximization) COP is a solution d to P that minimizes (maximizes) the value of $f(d)$.

CP-model

CP formulation:

$$
\begin{aligned}
\textit{variables}: \quad & \texttt{domain}(\texttt{y}_\texttt{i}) = \{1, \ldots, K\} && \forall i \in V \\
\textit{constraints}: \quad & y_i \neq y_j && \forall ij \in E(G) \\
& \texttt{alldifferent}(\{\texttt{y}_\texttt{i} \mid \texttt{i} \in \texttt{C}\}) && \forall C \in \mathcal{C}
\end{aligned}
$$

Propagation: An Example

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	(R)	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	(R)	B	(G)	R B	R G B	B	R G B
After *V=blue*	(R)	B	(G)	R	(B)		R G B

Figure 5.6 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

Outline

Constraint based Modelling

Can be done within the same framework of Constraint Programming.
See Constraint Based Local-Search (Hentenryck and Michel) [B4].

- Decide the variables.
  An assignment of these variables should identify a candidate solution or a candidate solution must be retrievable efficiently
  Must be linked to some Abstract Data Type (arrays, sets, permutations).

- Express the constraints on these variables

No restrictions are posed on the language in which the above two elements are expressed.

Outline

Search

- Backtracking (complete)
- Branch and Bound (complete)
- Local search (incomplete)

Outline

Summary

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

Outlook

Next Time:

- Construction Heuristics
- High level description of Local Search
- Solver Systems
- Setting up the Working Environment

In preparation:

- Lecture notes
- Revise basic concepts in algorithmics (see slides available at the web page and complement them with Cormen, Leiserson, Rivest and Stein. *Introduction to algorithms*. 2001)
- Reading material
- Obligatory assignment 1 launched