DM841

Discrete Optimization

Lecture 1

Course Introduction Discrete Optimization

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Overview

1. Course Introduction

2. Discrete Optimization
   Combinatorial Problems

Outline

Schedule and Material

- Communication media
  - Course Public Webpage (WWW) $\Leftrightarrow$ BlackBoard (BB)
    (link from http://www.imada.sdu.dk/~marco/DM841/)
  - Announcements in BlackBoard
  - Course Documents (for photocopies) in (BB)
  - Discussion Board (anonymous) in (BB)
  - Personal email

- Class schedule:
  - See course web page (get the ical-feed).
  - Week 37: 4 classes. Move Friday to Tuesday?

- Working load:
  - Intro phase (Introfase): 48 hours, 24 classes
  - Skills training phase (Træningsfase): 20 hours, 10 classes
  - Study phase: (Studiefase) 30 hours

  We have 39 classes scheduled.

Course Organization

Basically two Parts (old DM811+DM826):

Part I: ($\approx$ 12 classes)

- Local Search
- Metaheuristics
- EasyLocal
- Efficiency issues
- Experimental Analysis

Part II: ($\approx$ 12 classes)

- Modeling in CP
- Local Consistency
- Constraint Propagation
- Search
- Symmetry Breaking

Evaluation

- Obligatory Assignments:
  Part I:
  Two preparation assignments with pass/fail
  One midterm with 7-grade scale + external censor
  Part II:
  One/Two preparation assignments with pass/fail
  One final assignment with 7-grade scale + external censor

- All assignments must be passed.

- Final grade is average of midterm and final assignments.

- Preparation assignments can be prepared in pairs but individual submission $\rightsquigarrow$ Feedback

- Midterm and final assignments are individual and communication not allowed.

Content of the Assignments

Part I:

- Algorithm design
- Implementation (deliverable and checkable source code)
- Written description
- (Analytical) and experimental analysis
- Performance counts!

Part II:

- Manual computations
- Modeling
- Implementation (deliverable and checkable source code)
- Written description
- (Analytical) and experimental analysis
- Performance counts!

Problem Solving Contest, web platform for submission

Literature

- Part I (on Local Search):

  HM P.V. Hentenryck and L. Michel. Constraint-Based Local Search. The MIT Press, Cambridge, USA, 2005. (In BlackBoard)

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

  HS H. Hoos and T. Stuetzle, Stochastic Local Search: Foundations and Applications, 2005, Morgan Kaufmann

- Part II (on Constraint Programming):

  RBW F. Rossi, P. van Beek and T. Walsh (ed.), Handbook of Constraint Programming, Elsevier, 2006

  STL C. Schulte, G. Tack, M.Z. Lagerkvist, Modelling and Programming with Gecode 2013

- Other sources:

  RN S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. (Part II, chp. 3,4,6). Third Edition. Prentice Hall, 2010.
  - https://class.coursera.org/optimization-001

- Photocopies from Course Documents left menu of BlackBoard
- Lecture slides

Applied Character

Optimization problems are very challenging, seldom solvable exactly in polynomial time and no single approach is likely to be effective on all problems.
Solving optimization problems remains a very experimental endeavor: what will or will not work in practice is hard to predict. [HM]

Hence the course has applied character:

- We will learn the theory (mostly in Part II)
- but also implement some solvers $\rightsquigarrow$ programming in C++
- We will learn to analyze the results

Desired Active Participation

Be prepared for:

- Problem solving in class
- Hands on experience with programming
- Experimental analysis of performance
- Discussion on exercise sheets

These activities will be announced the class before

They require study phase (= work outside the classes)

Software

Here, we will use *free* and open-source software:

- Local Search: EasyLocal++ (C++) – GNU General Public License
- Constraint Programming: Gecode (C++) – MIT license
- Experimental Analysis: R – The R project

Many others, some commercial

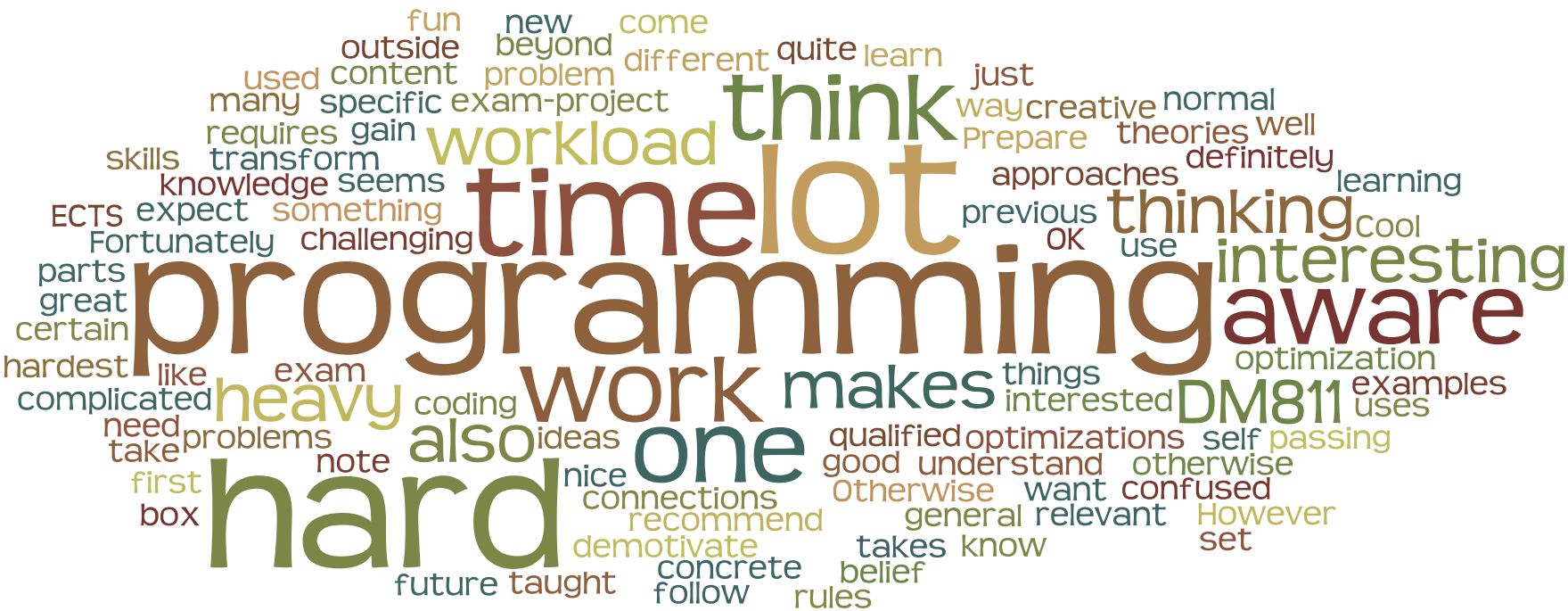
Knowledge in Programming and Algorithm and Data Structures is assumed.
Regarding C Language and pointers, what is the stand of the class?

Former students' feedback (1/2)

On the course:

- the course builds 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
- the assignments are really an important preparation to the final projects
- Gruppearbejdet og praktiske eksempler var gode og brugtbare
- The course was intellectually stimulating
- It is not always easy to know the standard of work expected
- Better with separation between submission of code and report

Word cloud


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

- 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
- bad descriptions
- mistaken data aggregation in instance set analysis.

Good/bad examples and rubric of comments will be made available

Outline

Outline

Discrete and Combinatorial Optimization

- Discrete optimization emphasizes the difference to continuous optimization, solutions are described by integer numbers or discrete structures
- Combinatorial optimization is a subset of discrete optimization.
- Combinatorial optimization is the study of the ways discrete structures (eg, graphs) can be selected/arranged/combined: Finding an optimal object from a finite set of objects.
- Discrete/Combinatorial Optimization involves finding a way to efficiently allocate resources in mathematically formulated problems.

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
- auction winner determination
- portfolio selection
- ...

Combinatorial Problems

Simplified models are often used to formalize real life problems

- coloring graphs (GCP)
- finding models of propositional formulae (SAT)
- finding variable assignment that satisfy constraints (CSP)
- 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

Elements of Combinatorial Problems

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.

Note, in this course:
Candidate solutions are combinations of objects or solution components that need not satisfy all given conditions.

Feasible solutions are candidate solutions that satisfy all given conditions.

Optimal Solutions are feasible solutions that maximize or minimize some criterion or objective function.

Approximate solutions are feasible candidate solutions that are not optimal.

Combinatorial Problems: Examples

Selecting

Grouping:

Given a finite set $N = \{1, \ldots, n\}$, weights c_j for each $j \in N$, and a set $\mathcal{F}$ of feasible subsets of N, find a minimum weight feasible subset of N, ie,

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} c_j \mid S \in \mathcal{F} \right\}$$

- **candidate solution:** one of the $2^{|N|}$ possible subsets of N.
- **(Optimal) solution:** the feasible subset of minimal cost

Combinatorial Problems: Examples

Ordering

Ordering:

Traveling Salesman Problem

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

- **candidate solution:** one of the $(n-1)!$ possible sequences of points to visit one directly after the other.
- **(Optimal) 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 exist
- 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.

- 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.
- Many optimization problems have an objective function as well as constraints (= logical conditions) that solutions must satisfy.
- **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.

Summary

1. Course Introduction

2. Discrete Optimization
   Combinatorial Problems