DM841

Discrete Optimization

Lecture 1
Course Introduction
Constraint Programming

Combination

Simplification

Contradiction

Redundancy

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Motivation

2. Course Organization

Outline

1. Motivation

2. Course Organization

Main Aim of the Course

To enable the student to solve discrete optimization problems that arise in practical applications

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.

Discrete Optimization Problems

Discrete Optimization problems

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

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

Discrete Optimization Problems

Simplified models are often used to formalize real life problems

- finding models of propositional formulae (SAT)
- finding variable assignment that satisfy constraints (CSP)
- partitioning graphs or digraphs
- partitioning, packing, covering sets
- finding shortest/cheapest round trips (TSP)
- coloring graphs (GCP)
- 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.

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 but good in some sense.

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
- but also implement some solvers $\rightsquigarrow$ programming in C++
- We will learn how to analyze the experimental results

Expected prerequisites

Students taking the course are expected to:

- Have knowledge of linear and integer programming
- Be able to use algorithms and data structures
- Be able to assess the complexity of the algorithms with respect to runtime and space consumption
- Be able to program

Outline

1. Motivation

2. Course Organization

Course Organization

Two Parts:

Part I: Constraint Programming (CP)
Part II: Heuristics

Part I: ($\approx$ 12 classes)

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

Part II: ($\approx$ 12 classes)

- Local Search
- Metaheuristics
- Implementation Framework
- Efficiency issues
- Experimental Analysis

Schedule

- Class schedule:
  - See course web page.
  - `mitsdu.sdu.dk`

- Working load:
  - Intro phase (Introfase): 48 hours, 24 classes
  - Skills training phase (Træningsfase): 40 hours, 20 classes
  - Study phase: (Studiefase) ?? hours

  We have 42 classes scheduled.

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 weighted(?) average of midterm and final assignments.

- Preparation assignments can be prepared in pairs but individual submission ⇝ Feedback

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

Learning Objectives

For a top perfomance the student must demonstrate ability to:

- model a problem similar in nature to the ones seen in the course within the framework of constraint programming and of local search
- argue about the different modeling choices arising from the theory behind the components of constraint programming, including global constraints, propagators, search and branching schemes.
- develop a solution prototype in a constraint programming system
- design specialized versions of general purpose heuristics: construction heuristics and local search
- develop a solution prototype in a local search framework
- undertake an experimental analysis, report the results and draw sound conclusions based on them
- describe the work done in an appropriate language including pseudocode

Content of the Graded Assignments

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

Web submission with automatic check, execution and comparison.

Competences wrt Degree

- plan and carry out scientific projects at the high professional level including managing work and development situations that are complex, unpredictable and require new solutions
- describe, analyze and solve advanced computational problems using the learned models
- analyze the advantages and disadvantages of various algorithms, especially in terms of resource consumptions
- elucidate the hypotheses of qualified theoretical background and critically evaluate own and others' research and scientific models
- develop new variants of the methods learned where the specific problem requires
- communicate through a written report research based knowledge and discuss professional and scientific problems with peers

Communication media

- Public Web Page [WWW] ⇔ BlackBoard e-learn.sdu.dk [BB] (link from http://www.imada.sdu.dk/~marco/DM841/)
- Announcements in BlackBoard
- Course Documents in [BB] (unless linked from [WWW])
- Discussion Board (anonymous) in [BB]
- Personal email marco@imada.sdu.dk
- Office visits
- (A-bit-earlier-than) Mid term evaluation in class

Literature

- Part I (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 2015

- Part II (on Local Search):

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

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

- Other sources: articles, slides, lecture notes

Software

Under development:
http://www.minizinc.org/challenge2014/results2014.html

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

- Constraint Programming: Gecode (C++) – MIT license
- Local Search: C++
- Experimental Analysis: R – The R project

Many others, some commercial

Knowledge in Programming and Algorithm and Data Structures is assumed.
C/C++ Language

Agreement for the Exercise Sessions

- Read the text before meeting at the class
- If you encounter difficulties then come to my office or write me an email, or take note of the question and bring it in class
- The meaning with the exercise classes is for you to get feedback on your level, not to deliver new material
- All questions and comments are welcome
- There is not stupid question and we all learn from mistakes.
- If you cannot resist to use internet to check emails and browse in Facebook, then do it outside of this room, so that you do not disturb the others.
- I can ask questions to everybody and it is not to punish someone. You can well say pass.

Class format

Be prepared for:

- Flipped classes: learn content at home, engage with material in class
- Problem solving in class
- Hands on experience with programming
- Experimental analysis of performance
- Discussion on exercises for home

These activities will be announced

They require study phase (= work outside the classes)

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
- Group work and practical examples were good and usable
- The course was intellectually stimulating
- It is not always easy to know the standard of work expected
  assignments were too open
- Better with separation between submission of code and report

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

You

- Whole course or a part?
- Background
  education line
  programming skills
  DM559/DM545, integer and linear
- Expectations