DM841 (10 ECTS)
Heuristics and Constraint Programming for Discrete Optimization

Marco Chiarandini, Asc Prof, IMADA
https://imada.sdu.dk/u/march
Video (14 min): https://imada.sdu.dk/u/march/Videos/dm841.mp4

Course Formalities

Target students:	• computer science • applied math • math and economics • data science
When:	stretched over a full semester of the Bachelor or Master
Prerequisites:	Programming (Java or Python)

Decision Problems with Discrete Variables

Many decision problems (logistics, production planning, timetabling, etc.) aiming at an **optimal use of resources** can be formulated as constrained discrete optimization problems.

Example: Social Golfer Problem (Combinatorial Design):

- 9 golfers: {1, 2, 3, 4, 5, 6, 7, 8, 9}
- wish to play in groups of 3 players in 4 days
- such that no golfer plays in the same group with any other golfer more than just once.

Is it possible?

	Group 1	Group 2	Group 3
Day 0	? ? ?	? ? ?	? ? ?
Day 1	? ? ?	? ? ?	? ? ?
Day 2	? ? ?	? ? ?	? ? ?
Day 3	? ? ?	? ? ?	? ? ?

Solution Paradigms

- Dedicated algorithms
  (eg.: enumeration, branch and bound, dynamic programming)
- Integer Linear Programming (DM871/DM545)
- Constraint Programming:
  representation (modeling) + reasoning (search + inference)
- Heuristics & Metaheuristics
  representation (modeling) + reasoning (search)
- Other (encode as SAT, SMT, etc.)

Constraint Programming

Constraint programming aims at reducing the cost of developing solvers for combinatorial problems through extensive reuse of code for pruning the search space by reasoning at the level of constraints. Problems can be solved by just specifying a model, eg, in MiniZinc.

Integer variables:
`assign[i,j]` variable whose value is from the domain $\{1, 2, 3\}$
Constraints:
C1: each group has exactly groupSize players
C2: each pair of players only meets once

```
int: golfers = 9;
int: groupSize = 3;
int: days = 4;
int: groups = golfers/groupSize;

set of int: Golfer = 1..golfers;
set of int: Day = 1..days;
set of int: Group = 1..groups;

array[Golfer, Day] of var Group: assign; % Variables

constraint
  % C1: Each group has exactly groupSize players
  forall (gr in Group, d in Day)( % c1
   sum (g in Golfer) (bool2int(assign[g,d] = gr)) = groupSize
  )
  /\
  % C2: Each pair of players only meets at most once
  forall (g1, g2 in Golfer, d1, d2 in Day  where g1 != g2 /\ d1 != d2) (
   (bool2int(assign[g1,d1] = assign[g2,d1]) + bool2int(assign[g1,d2] = assign[g2,d2])) <=1
  );

solve :: int_search([assign[i,j] | i in Golfer, j in Day ],
                    first_fail, indomain_min, complete)  satisfy;
```

Groups

	Day 0	Day 1	Day 2	Day 3
Golfer 0	1	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 1	1	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 2	1	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 3	{2,3}	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 4	{2,3}	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 5	{2,3}	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 6	{2,3}	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 7	{2,3}	{1,2,3}	{1,2,3}	{1,2,3}
Golfer 8	{2,3}	{1,2,3}	{1,2,3}	{1,2,3}

Heuristics

Heuristic algorithms: compute, efficiently, good solutions to a problem (without caring for theoretical guarantees on running time and approximation quality).

Modeling

- Solution representation and tentative solution
- Constraints:
  → implicit
  → soft
- Evaluation function

	Group 1	Group 2	Group 3
Day 0	0 1 2	3 4 5	6 7 8
Day 1	**0 4** 6	1 **3 7**	2 5 8
Day 2	**0 4** 8	1 5 6	2 **3 7**
Day 3	0 5 7	1 3 8	2 4 6

Solution approach: search by trial and error or other nature inspired processes
It requires ad hoc implementations (programming)

Contents

Constraint Programming:

- Modelling applications
  Integer variables, set variables, constraints
- Constraint reasoning
  Consistency levels
- Filtering algorithms
  Alldifferent, cardinality, regular expressions, etc.
- Search:
  Backtracking, Strategies
- Symmetry breaking
- Restart techniques
- CP Systems: MiniZinc

Heuristics

- Construction heuristics
- Local search
- Metaheuristics
  → Simulated annealing
  → Iterated local search
  → Tabu search
  → Variable neighborhood search
  → Evolutionary algorithms
  → Ant colony optimization
- Programming in Python

Aims & Format

Aims:

- learning to solve discrete optimization problems with concrete real-life applications
- modeling in constraint programming
- design and implement heuristic algorithms
- assessing the solution approaches deployed
- describing in appropriate language the work done

Format:

- Full semester course
- Two classes per week plus one exercise class including hands on practice
- Course material: slides, articles and chapters from books

Exam Form

Five obligatory assignments:

- individual
- deliverables: program + short written report
- Two of the five are graded with internal censor, final grade given by weighted average of these two