

DM811  
Heuristics for Combinatorial Optimization

Lecture 3  
**Construction Heuristics and Metaheuristics**

Marco Chiarandini

Department of Mathematics & Computer Science  
University of Southern Denmark

1. Work Environment  
Organization

2. Complete Search Methods  
Constraint Satisfaction Problems

2

What will you need during the project? How will you organize it? How will you make things work together?

1. Work Environment  
Organization

2. Complete Search Methods  
Constraint Satisfaction Problems

- `src/` `code` that implements the algorithm (likely, several versions)
- `bin/` place where to put your executables
- `data/` `input`: Instances for the algorithm, parameters to guide the algorithm, instructions for reporting.
- `res/` `output`: The result, the performance measurements
- `R/` `Analysis tools`: statistics, data analysis, visualization
- `doc/` `journal`: A record of your experiments and findings.
- `log/` other log files produced by the run of the algorithm

## Input controls on command line

```
comet queens.co -i instance.in -o output.sol -l run.log > data.out
```

## Output on stdout, self-describing

```
#stat instance.in 30 90
seed: 9897868
Parameter1: 30
Parameter2: A
Read instance. Time: 0.016001
begin try 1
best 0 col 22 time 0.004000 iter 0 par_iter 0
best 3 col 21 time 0.004000 iter 0 par_iter 0
best 1 col 21 time 0.004000 iter 0 par_iter 0
best 0 col 21 time 0.004000 iter 1 par_iter 1
best 6 col 20 time 0.004000 iter 3 par_iter 1
best 4 col 20 time 0.004000 iter 4 par_iter 2
best 2 col 20 time 0.004000 iter 6 par_iter 4
exit iter 7 time 1.000062
end try 1
```

If a single program that implements many heuristics

- re-compile for new versions but take old versions with a journal in archive.
- use command line parameters to choose among the heuristics
- C: getopt, getopt\_long, opag (option parser generator)  
Java: package `org.apache.commons.cli`  
Comet: see example provided `loadDIMACS.co`

```
comet queens.co -i instance.in -o output.sol -l run.log -solver 2-opt > data.out
```

- use identifying labels in naming file outputs  
Example:

```
c0010.i0002.t0001.s02010.log
```

7

8

- You will need:  
multiple runs, multiple instances, multiple classes and multiple algorithms.  
Arrange this outside of your program: ➔ unix scripts (eg, bash one line program, perl, php)
- Parse outputfiles:  
Example  

```
grep #stat | cut -f 2 -d " "
```

  
See <http://www.gnu.org/software/coreutils/manual/> for shell tools.
- Data in form of matrix or data frame goes directly into R imported by `read.table()`, untouched by human hands!

```
alg instance      run sol time
ROS le450_15a.col 3 21 0.00267
ROS le450_15b.col 3 21 0
ROS le450_15d.col 3 31 0.00267
RLF le450_15a.col 3 17 0.00533
RLF le450_15b.col 3 16 0.008
...
```

Visualization helps understanding

- Problem visualization (graphviz, igrph)
- Algorithm animation: (comet visualize)
- Results visualization: recommended R (more on this later)

9

10

- Check the correctness of your solutions many times
- Plot the development of
  - best visited solution quality
  - current solution qualityover time and compare with other features of the algorithm.

11

### Planning

Release planning creates the schedule • Make frequent small releases • The project is divided into iterations

### Designing

Simplicity • **No functionality is added early** • Refactor: eliminate unused functionality and redundancy

### Coding

Code must be written to agreed standards • **Code the unit test first** • All production code is pair programmed • **Leave optimization till last** • No overtime

### Testing

All code must have unit tests • All code must pass all unit tests before it can be released • When a bug is found tests are created

13

- Profile time consumption per program components
  - under Linux: gprof
    1. add flag `-pg` in compilation
    2. run the program
    3. `gprof gmon.out > a.txt`
  - Java VM profilers (plugin for eclipse)

12

1. Work Environment  
Organization

2. Complete Search Methods  
Constraint Satisfaction Problems

14

## 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  $x_1, \dots, x_n$ , together with an objective function  $f : D(x_1) \times \dots \times D(x_n) \rightarrow Q$  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  $f(d)$ .

- **initial state**: the empty assignment in which all variables are unassigned
- **successor function**: a **value** can be assigned to any unassigned **variable**, provided that it does not conflict with previously assigned variables
- **goal test**: the current assignment is complete
- **path cost**: a constant cost

Types of problems:

- Assignment
- Sequencing
- Subset
- Routing
- ...

15

16

## Uninformed

### Search Space

tree with branching factor at the top level  $nd$   
at the next level  $(n-1)d$ .

The tree has  $n! \cdot d^n$  even if only  $d^n$  possible complete assignments.

## Informed

- CSP is **commutative** in the order of application of any given set of action. (we reach same partial solution regardless of the order)
- Hence generate successors by considering possible assignments for only a single variable at each node in the search tree.
- look-ahead, best first, etc.

## Backtracking search

*depth-first search* that chooses one variable at a time and backtracks when a variable has no legal values left to assign.

**function** BACKTRACKING-SEARCH(*csp*) **returns** a solution, or failure  
**return** RECURSIVE-BACKTRACKING( $\{\}$ , *csp*)

**function** RECURSIVE-BACKTRACKING(*assignment*, *csp*) **returns** a solution, or failure  
**if** *assignment* is complete **then return** *assignment*  
*var*  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*)  
**for each** *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**  
    **if** *value* is consistent with *assignment* according to CONSTRAINTS[*csp*] **then**  
        add {*var* = *value*} to *assignment*  
        *result*  $\leftarrow$  RECURSIVE-BACKTRACKING(*assignment*, *csp*)  
        **if** *result*  $\neq$  failure **then return** *result*  
        remove {*var* = *value*} from *assignment*  
**return** failure

18

20

- No need to copy solutions all the times but rather extensions and undo extensions
- Since CSP is standard then the alg is also standard and can use general purpose algorithms for initial state, successor function and goal test.
- Backtracking is **uninformed and complete**. Other search algorithms may use **information** in form of heuristics.

21

Decisions in general purpose methods:

- 1) Which variable should we assign next, and in what order should its values be tried?
- 2) What are the implications of the current variable assignments for the other unassigned variables?
- 3) When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Search (1) + Inference (2) + Backtracking (3) = **Constraint Programming**

In the general case, at point 1) we use heuristic rules.

If we do not backtrack (point 3) then we have a **construction heuristic**.

22