Department of Mathematics and Computer Science
University of Southern Denmark, Odense

January 10, 2013

Marco Chiarandini

# DM811 - Heuristics for Combinatorial Optimization

## Exam Project, Fall 2012, Reexam

---

**Remark 1** The project is carried out individually and it is not allowed to collaborate.

**Remark 2** The project consists of algorithm design, implementation, experimentation and written report.

The evaluation of the project is based on the report. However, a program that implements the best algorithm described in the report must also be submitted. The program will be used to verify the correctness of the results presented. The report may be written in Danish or in English.

**Remark 3** Corrections or updates to the project description, if any, will be published on the course web page and will be announced by email to the addresses available in the BlackBoard system. In any case, it remains students' responsibility to check for new announcements.

**Remark 4** *Submission.* An archive containing the electronic version of the written report and the source code of the program must be handed in through the BlackBoard system with deadline

**Saturday, January 26, 2013 at 12:00.**

The submission procedure is the following:

- choose the course DM811 in BlackBoard,
- choose "SDU Assignment" in the menu on the left,
- fill the form and conclude with submit,
- print and conserve the receipt (there will be a receipt also per email).

See Appendix D for details on how to organize the electronic archive.

In addition to the electronic submission you must deposit two printed copies of your report at the teacher's mailbox in the secretary office.

Reports and codes handed in after the deadline will generally not be accepted. System failures, illness, etc. will not automatically give extra time.

**Remark 5** Write your name, your CPR number and your user ID as it appeared during the graph coloring assignments in the front page of the report.

**Remark 6** Make sure you have read the whole document before you start to work.

**Remark 7** The evaluations of the exams will be sent to the study office not later than three weeks after the deadline has expired.

---

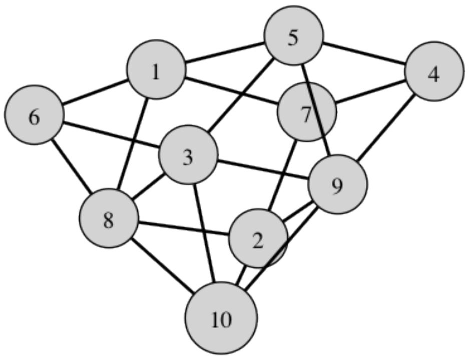

Figure 1: A graph $G = (V, E)$ with vertices $V = \{1, 2, ..., 10\}$ and weights $\{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

## 1 The Problem

Let $G = (V, E)$ be an undirected graph with a set $V$ of vertices and a set $E$ of edges. As usual, we set $n = |V|$ and $m = |E|$. A *clique* is a subset $C \subseteq V$ composed of pairwise adjacent vertices, that is, $\{v, w\} \in E$ for all $v, w \in C$. The size of a largest clique in $G$ is called *clique number*. A maximal clique is any clique in a graph that cannot be extended any further. Note that the size of a maximal clique can be smaller than the largest clique in the graph.

A *stable set* is a subset $S \subseteq V$ composed of pairwise non-adjacent vertices, that is, $\{v, w\} \in E$ for all $v, w \in S$. The maximum size of a stable set in $G$ is called *stability number*. Cliques are stable sets in the complement $\bar{G} := (V, \{\{v, w\} \in V \times V : \{v, w \notin E, v \neq w\}\})$.

Let $\omega$ be a function that assigns a positive integer to each vertex of the graph, that is, $\omega : V \rightarrow \mathbf{Z}^+$. The Maximum-Weight Clique Problem (MWCLP) asks to find the clique of maximum total weight.

Consider the graph in Figure 1. It has the following 8 cliques of maximal size:

```
size=3, weight=180:   4 5 9
size=3, weight=170:   3 6 8
size=3, weight=210:   3 8 10
size=3, weight=220:   3 9 10
size=3, weight=170:   3 5 9
size=3, weight=200:   2 8 10
size=3, weight=210:   2 9 10
size=3, weight=150:   1 6 8
```

Hence, the clique number of the graph is 3. The maximum-weight clique is made of vertices 3, 9, 10 and has total weight of $30 + 90 + 100 = 220$. The weighted clique number of the graph is 220.

## 2 Project Requirements

The aim of the project is to study heuristic algorithms for MWCLP on arbitrary test instances. The test instances are made available for the assignment at `http://www.imada.sdu.dk/~marco/DM811/Projects/instances.tgz` and are listed in Table 1. The instance marco10.col used in Figure 1 is also included for testing purposes. A solution checker will be made available during the first days of January. The format of the instances as well as the requirements for the solution are in the appendix.

All the following points must be addressed to pass the exam:

1. Design and implement one or more construction heuristics

| instance | $n$ | $m$ | Best known |
|---|---|---|---|
| DSJC1000.1.mwclq.3915.dimacs | 998 | 448010 | 2028496 |
| DSJC1000.5.mwclq.dimacs | 1000 | 249674 | |
| DSJC1000.9.mwclq.dimacs | 1000 | 50051 | 2147482 |
| DSJC125.1.mwclq.dimacs | 123 | 6778 | |
| DSJC125.5.mwclq.dimacs | 125 | 3859 | |
| DSJC125.9.mwclq.dimacs | 125 | 789 | |
| DSJC250.1.mwclq.dimacs | 241 | 25840 | |
| DSJC250.5.mwclq.dimacs | 250 | 15457 | |
| DSJC250.9.mwclq.dimacs | 250 | 3228 | |
| DSJC500.1.mwclq.117.dimacs | 494 | 109554 | 4125294 |
| DSJC500.5.mwclq.dimacs | 500 | 62126 | |
| DSJC500.9.mwclq.dimacs | 500 | 12313 | |

Table 1: The instances to be solved in the assignment.

| instance | weight |
|---|---|
| DSJC1000.1.mwclq.3915.dimacs | |
| DSJC1000.5.mwclq.dimacs | |
| DSJC1000.9.mwclq.dimacs | |
| DSJC125.1.mwclq.dimacs | |
| DSJC125.5.mwclq.dimacs | |
| DSJC125.9.mwclq.dimacs | |
| DSJC250.1.mwclq.dimacs | |
| DSJC250.5.mwclq.dimacs | |
| DSJC250.9.mwclq.dimacs | |
| DSJC500.1.mwclq.117.dimacs | |
| DSJC500.5.mwclq.dimacs | |
| DSJC500.9.mwclq.dimacs | |

Table 2: Fill in the table with numerical results

2. Design and implement one or more stochastic local search or metaheuristics algorithms.
3. Carry out an experimental analysis with the following goals:
   - tune the parameters of the algorithms designed at the previous points.
   - compare the algorithms designed at the previous two points (hence there must be more than one algorithm coming out from point 1) and 2)
4. Report the results of the best algorithm you found in the previous point in a table like Table 2. Use **one minute** as time limit.
5. Describe the work in the previous points in a report commenting the experimental results and drawing sound conclusions.

In the experimental assessment of any algorithm limit the maximum computation to one minute.

## 3 Remarks

**Remark 1** For each point above a description must be provided in the report of the work undertaken. In particular for the best algorithm arising from the experimental analysis enough details must be provided in order to guarantee the reproducibility of the algorithm from the report alone (i.e., without need for looking at the source code). It is important to give account of the computational cost of the operations in the local search.

**Remark 2** The total length of the report should not be less than 5 pages and not be more than 12 pages, appendix included (lengths apply to font size of 11pt and 3cm margins). Although these bounds are not strict, their violation is highly discouraged. In the description of the algorithms, it is allowed (and encouraged) to use short algorithmic sketches in form of pseudo-code but not to include program codes.

**Remark 3** This is a list of factors that will be taken into account in the evaluation:

- quality of the final results;
- level of detail of the study;
- complexity and originality of the approaches chosen;
- organization of experiments which guarantee reproducibility of conclusions;
- clarity of the report;
- effective use of graphics in the presentation of experimental results.

**Remark 4** In the project requirements of Sec. 2 the words “one or more algorithms” do NOT imply “the more algorithms, the better the final grade”. A few, well thought algorithms are better in this sense than many naive ones!

## Appendix A Instance Format

All graphs are in DIMACS format, which consists of a file where each line begins with a letter that defines the content of the line. The legal lines are:

- `c` Comment: remainder of line ignored.
- `p` Problem: is in the form `p type n m` where `n` is the number of vertices (to be numbered $1..n$) and `m` the number of edges. The field `type` can be any word and it must be ignored.
- `n` Vertex: is in the form `n v w` where `v` is the identifier of the vertex and `w` its weight.
- `e` Edge: is in the form `e n1 n2 d` where `n1` and `n2` are the endpoints of the edge and `d` is a parameter that must be ignored.

Note that each edge can be written twice and that **loops, that is, edges with head and tail on the same vertex must be ignored**.

## Appendix B Solution Format

In order to check the validity of the results reported, the program submitted must output when finishing the best solution found during its execution in a file with extension `.sln`. The file must be in text format and contain the selected clique by listing a vertex per row.

For example, the solution of Figure 1 would be printed as:

```
$> cat marco10.sln
3
9
10
```

## Appendix C Solution Validator

A program to check the validity of a given solution will be made available at the course web page. The program runs on the Linux machines of the IMADA terminal room.

To verify a solution type from the command line something like the following:

```
make
test_col -i marco10.col -s marco10.sln -p 7
```

## Appendix D Handing in Electronically

Your work must be handed in electronically via BlackBoard. In addition two printed copies of the report must be deposited at the teacher's mailbox in the secretary office. The official receipt will be obtained at the BlackBoard submission.

This section describes how you must organize your electronic submission.

Main directory:

```
    userID/
```

where `userID` is your login at SDU email

```
userID/README
userID/Makefile
userID/src/
userID/bin/
userID/res/
userID/doc/
```

The directory `doc` contains a pdf version of your report. The file README provides instructions for compilation of the program. The directory `src` contains the sources which may be in C, C++, Java or other languages. If needed a Makefile can be included either in the root directory or in `src`. After compilation the executable must be placed in `bin`. For java programs, a jar package can also be created via Makefile.

Programs must work on IMADA's computers under Linux environment and with the compilers and other applications present on IMADA's computers. Students are free to develop their program at home, but it is their own responsibility to transfer the program to IMADA's system and make the necessary adjustments such that it works at IMADA.[1]

The executable must be called `mwclq`. It must execute from command line by typing in the directory `userID/bin/mwclq`:

```
mwclq -i INSTANCE -s SEED -c OUTPUT -t TIME
```

where beside the flag to indicate the file of the instance it is:

- `-t TIME` the time limit in seconds;
- `-s SEED` the random seed;
- `-c OUTPUT` the file name where the solution is written.

For example:

```
mwclq -i marco10.col -c marco10.sln -t 120 -s 1 > marco10.log
```

will run the program on the instance marco10.col for 120 seconds with random seed 1 and write the solution in the file `marco10.sln`.

In its default mode, the program must run the best algorithm developed and must print on the standard output the weight of the clique found. Make sure that the one found is really a clique.

It is advisable to have a log of algorithm activities during the run. This can be achieved by printing further information on the standard error or in a file. A suggested format is to output a line whenever a new best solution is found containing at least the following pieces of information:

```
best 53 time 10.000000 iter 1000
```

All process times are the sum of user and system CPU time spent during the execution of a program as returned by the linux C library routine `getrusage`. If you are using the frameworks made available in the Graph Coloring Assignments continue to use the same functions for time checking.

[1]Past issue: the java compiler path is `/usr/local/bin/javac`; in C, any routine that uses subroutines from the math.c library should be compiled with the `-lm` flag – eg, `cc floor.c -lm`.