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

October 13, 2013
Marco Chiarandini

DM811 - Heuristics for Combinatorial Optimization

Final Assignment, Autumn 2013

Submission deadline: Thursday, October 24, 2013 at 12:00
(strict, no extension is possible)

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

1 Project Requirements

The aim of the final assignment is to revise and develop further the heuristic algorithms for solving the graph coloring problem that have been the object of the previous assignments. We refer to Assignment 0 for a definition of the problem.
The instances that will be used for the final comparison of results are instances of 4000 vertices and edge density of 0.1; 0.5; 0.9. These instances are huge and a binary format is needed to store them. You will need to update your framework to read this format. Read details on this in the Appendix B of this document.
A few test instances are made available here:

http://www.imada.sdu.dk/~marco/DM811/Color-4/instances/

The main challenge of this final assignment is the size of the instances. Make sure that a solution is returned in each run before the time limit is reached. The time limit is set to **120 seconds**. *Your program must be optimized and eventually redesigned to handled these huge instances.*

All the following points (that have been already addressed in the previous assignments) must be accomplished to pass the exam:

1. Design and implement two or more construction heuristics and show that they perform better than the solution obtained by a greedy algorithm that uses a random permutation of vertices.
2. Design and implement one or more stochastic local search or metaheuristic algorithms.
3. Carry out an experimental analysis where you compare and configure the algorithms from the previous two points. Describe the analysis in the report.
4. Describe the best construction heuristic and the best stochastic local search or metaheuristic algorithm in the report.

Instance	Construction heuristic		Metaheuristic
	# Colors	Seconds	# Colors
G-4000-0.1–00.col.b			
G-4000-0.1–01.col.b			
G-4000-0.5–00.col.b			
G-4000-0.5–01.col.b			
G-4000-0.9–00.col.b			
G-4000-0.9–01.col.b			

Table 1: The table shows the median results from 5 runs per instance of the best construction heuristics and the best metaheuristic designed. For the metaheuristic a time limit of 120 seconds is assumed.

5. Report the results of the best algorithms on the test instances made available in a table like Table 1.

6. On the basis of the experimental analysis decide which algorithm has to be executed when submitting your program. The programs will be run on a 64-bit machine with Ubuntu Linux, equivalent to those in the terminal room. A time limit of 120 seconds will be imposed.

2 Remarks

Remark 1 This final assignment is carried out individually and it is not allowed to collaborate.

Remark 2 The assignment is a revision and development of the previous 3 assignments. Therefore, it contains algorithm design, implementation, experimentation and written report.

Differently from the previous assignments, the submission will be graded with the 7-steps scale and external censorship. The evaluation of the project is based on the report and the results that will appear on the web page.

Hence, the submission consists of:

- an executable program that implements the best algorithm developed and returns valid solutions within the time limit;
- the source code of the executable program;
- a written report that may be written in Danish or in English.

It is expected that the feedback provided in the previous assignment is used in the revision of the work.

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 See Appendix A 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. Remember to collect and keep a receipt of the submission.

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 and your user ID as it appeared during the graph coloring assignments in the front page of the report.

Remark 6 The evaluation of the submissions will be communicated to the exam office not later than three weeks after the deadline has expired.

Remark 7 The report must contain enough details to guarantee the reproducibility of the algorithm and experiments from the report alone (i.e., without looking at the source code). It is important to give account of the computational cost of the operations in the construction heuristics and in the local search.

Remark 8 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 source codes.

Remark 9 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 that guarantees reproducibility of conclusions;
- clarity of the report;
- effective use of graphics in the presentation of experimental results.

Remark 10 In the project requirements of Sec. 1 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 Submission instructions

Appendix A is copied from Assignment 0, nothing has changed. The report must be placed in `doc/`.
Start early to test your submission. You can submit as many times as you wish within the deadline. Every new valid submission overwrites the previous submission.
Your archive must decompress as follows:

- `README`
- `bin/` where your executable called `gcp` must be
- `src/` the source files that implement the algorithm
- `Makefile` to compile the sources in `src` and puts the executables in `bin`.

Your program will NOT be recompiled; the executable file `gcp` in `bin/` will be used for the tests. `Makefile` and `src/` are required just for debugging purposes in special cases. Additionally, you are recommended to have the following content, which will however not be used.

- `data/` containing the test instances.
- `res/` containing your results, the performance measurements
- `r/` statistics, data analysis, visualization
- `doc/` a description of the algorithms.
- `log/` other log files produced by the run of the algorithm

The programs will run on a machine with ubuntu 12.04, hence you can log in on any machine of the terminal room, compile your program there, and make sure that it executes.
If your program is written in C or C++ you should compile with the `-lm` and `-static` flags.
If your program is written in Java then your bin directory must contain a jar file that you compiled on an IMADA machine and a shell executable file called `gcp` containing:

```
#!/bin/bash
HERE=‘dirname $0‘
java -jar $HERE/gcp.jar $@
```

See in Sec. B.2 how this file must be changed.
Python users can do the same but with python commands instead.

The executable must run with the following arguments:

- `-i filename` the instance file
- `-c filename` the file with the solution in the format described below
- `-s #` a random seed
- `-t #` an integer indicating the time limit in seconds.

for example:

```
gcp -i ../data/marco10.col -c marco10.sol -s 10 -t 60
```

All output must go in the standard output and in the solution file. Do not create other files. The files containing the solution must be named with the name of the instance (without `.col` extension) and with extension `.sol`. The files must be in ASCII format and consist of a single column of numbers representing the colors to each vertex. Each number indicates the color for the vertex corresponding to the line number. Both colors and vertices start at 1. For example:

```
$ cat marco10.sol
2
3
4
3
2
2
1
...
```

indicates that the vertex 1 receives color 2, vertex 2 receives color 3, etc.
Note that the upload and analysis system is still under test and therefore it may have to be fine tuned, hence be patient and contact the teacher if something is not working as it should.

Appendix B The new instance format

To read the new instances you need to update the framework made available at Assignment 0.

B.1 In C++

Follow these steps:

- Download the two files: `bin2asc.h` and `bin2asc.cpp` and put them in `src` with the other sources.
- In `Makefile`, add `bin2asc.o` among the object files
- In `Problem.h`, include `bin2asc.h`.
- In `Problem.cpp`, change the code that handles the case `BIN_FORMAT`. Precisely, you need to substitute this piece of code:

```
cout << "...read␣input␣in␣GZIP␣format..." << endl;
exit(1);
break;
```

with this:

```
cout << "...read␣input␣in␣BIN␣format..." << endl;
DIMACS_bin_format in(id);
num_of_vertices = in.Nr_vert;
adjacency_matrix = new unsigned char *[num_of_vertices];
Vertex v, w;
for (v = 0; v < num_of_vertices; v++) {
  adjacency_matrix[v] = new unsigned char[num_of_vertices];
  for (w = 0; w < num_of_vertices; w++)
    adjacency_matrix[v][w] = 0;
}
for (w = 0; w < in.Nr_vert; w++) {
  for (v = 0; v <= w; v++) {
    if (in.get_edge(w, v)) {
      //printf("e %d %d\n",w+1,v+1 );
      if (adjacency_matrix[v][w] == 0) {
        num_of_edges++;
      }
      adjacency_matrix[v][w] = adjacency_matrix[w][v] = 1;
    }
  }
}
assert(num_of_edges == in.Nr_edges);
break;
```

In case, you can also downloaded the updated framework: FrameworkC++.tgz.

B.2 In Java

A possibility is to translate the C++ files in Java. Alternatively you can use them as library via SWIG. This has already been prepared for you. You need to follow these steps:

- Download the archive swig.tgz uncompress and put the directory `swig` in the root of the FrameworkJava that was distributed with Assignment 0. Enter in the `swig` directory and compile the library via `make`. You need to compile the package on one of the IMADA machines. It has been tested on Linux and MacOS.

  If something goes wrong you can try to download the precompiled package for different architectures:

  - swig.x86_64.tgz
  - swig.x86.tgz
  - swig.darwin.x86_64.tgz

  You will need to include and link the first in the list for the execution on the dedicated machine. If you compile the library on a 32-bit Ubuntu Linux machine, the library will not work on a 64-bit architecture.

- In the `Makefile` inside `src`, add `../swig/bin2asc.jar` to the class path
- In `manifest` inside `meta/` add `../swig/bin2asc.jar` to `Class-Path`, thus the content of the file becomes:

```
Main-Class: GCP
Class-Path: . ../lib/commons-cli-1.2.jar ../swig/bin2asc.jar
```

- In the executable `bin/gcp` add `-Djava.library.path="${HERE}/../swig"`, thus its content becomes:

```
#!/bin/bash
HERE=‘dirname $0‘
java -Djava.library.path="${HERE}/../swig" -jar $HERE/gcp.jar $@
```

- You should now be ready to use the library from your `Problem.java`. To do this you must add inside the class `Problem`:

```
static {
   try {
     System.loadLibrary("bin2asc");
   } catch (UnsatisfiedLinkError e) {
     System.err.println("Native␣code␣library␣failed␣to␣load.␣See␣the␣chapter␣on␣Dynamic
         ␣Linking␣Problems␣in␣the␣SWIG␣Java␣documentation␣for␣help.\n" + e);
     System.exit(1);
   }
 }
```

and then use the library in the constructor of the class `Problem` for example as shown here:

```
 int format = 0;
 int BIN_FORMAT=1;
 int ASCII_FORMAT=2;

 if (filename.contains(".b"))
   format = BIN_FORMAT;
 else
   format = ASCII_FORMAT;

 if (format == BIN_FORMAT)
   {
     System.out.println("Reading␣in␣bin␣format");
     DIMACS_bin_format in = new DIMACS_bin_format(filename);
     //in.write_graph_DIMACS_ascii("tmp.col");
     numVertices = in.getNr_vert();
     numEdges = in.getNr_edges();
     colors = new int[numVertices];
     adjacency_list = (List<Integer>[])new List[numVertices];
     adj_matrix = new boolean[numVertices][numVertices];
     for(int i=0; i < numVertices; i++){
       adjacency_list[i] = new ArrayList<Integer>();
       for(int j=i; j < numVertices; j++){
         adj_matrix[i][j]=false;
         adj_matrix[j][i]=false;
       }
     }

     for (int u = 0; u < in.getNr_vert(); u++) {
```

```
      for (int v = 0; v < u; v++) {
        //System.out.println(u+" "+v+"-"+(int) in.get_edge(u, v)+"-");
        if (((int) in.get_edge(u, v)==1))
        {
          //printf("e %d %d\n",w+1,v+1 );
          adjacency_list[u].add(v);
          adjacency_list[v].add(u);
          adj_matrix[u][v]=true;
          adj_matrix[v][u]=true;
        }
      }
    }
  }
  else if (format==ASCII_FORMAT)
  {
    System.out.println("Reading␣in␣ASCII␣format");
      // The old source code of the constructor
  }
```

In case, you can also downloaded the updated framework: FrameworkJava.tgz.