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

September 3, 2013
Marco Chiarandini

DM811 - Heuristics for Combinatorial Optimization

Assignment 0, Autumn 2013

Submission deadline: Monday, September 09, 2013 at 12:00.

This a series of (1+3) obligatory assignments that has the goal of training and providing feedback in preparation of the final submission that will be graded. Throughout the assignments you will set up a developing environment, gain practical experience on the methods discussed in class, try your own ideas on a problem different from those discussed in class, and carry out the analysis of experimental results. The content of the assignments will also provide elements for discussion in class. You are encouraged to search feedback among your peers, and ask questions in class related to the assignment. Working in pairs is allowed, but since the final submission will be individual in order to prepare your tools, you are recommended to develop source code on your own (be aware that not only the idea but also how it is implemented is relevant for the final performance of an algorithm – and performance counts in this course!). The submission of the obligatory assignments is individual.

There will be 1+3 obligatory assignments. All assignments 1 to 3 must be passed in order to be admitted to the final exam. Assignment 0 has a slightly different character with respect to the others. It will not be strictly obligatory and a missed submission to assignment 0 will not compromise the possibility to the to take part in the other assignments and in the final submission. Note however that Assignments 1 to 3 build on Assignment 0 hence it is very likely that the work for this assignment must be done at some point.

The assignments are focused on the well known Graph Coloring Problem.

1 Graph Coloring Problem

The GRAPH (VERTEX) COLORING PROBLEM (GCP) asks to find an assignment of colors to the vertices of a graph in such a way that no adjacent vertices receive the same color. See Figure 1 for an example. Graph coloring problems arise in many real life applications like register allocation, air traffic flow management, frequency assignment, light wavelengths assignment in optical networks and timetabling. More formally, let $G = (V, E)$ be an undirected graph, with V being the set of $|V| = n$ vertices and E being the set of edges. A k-coloring of G is a mapping $\varphi : V \to \Gamma$, where $\Gamma = 1, 2, \ldots, k$ is a set of $|\Gamma| = k$ integers, each one representing a color. A k-coloring is *feasible* or *proper* if for all $[u, v] \in E$ it holds that $\varphi(u) \neq \varphi(v)$; otherwise it is *infeasible*. If for some $[u, v] \in E$ it is $\varphi(u) = \varphi(v)$, the vertices u and v are in *conflict*. A feasible k-coloring in which some vertices are uncolored is said to be a *partial k-coloring*.

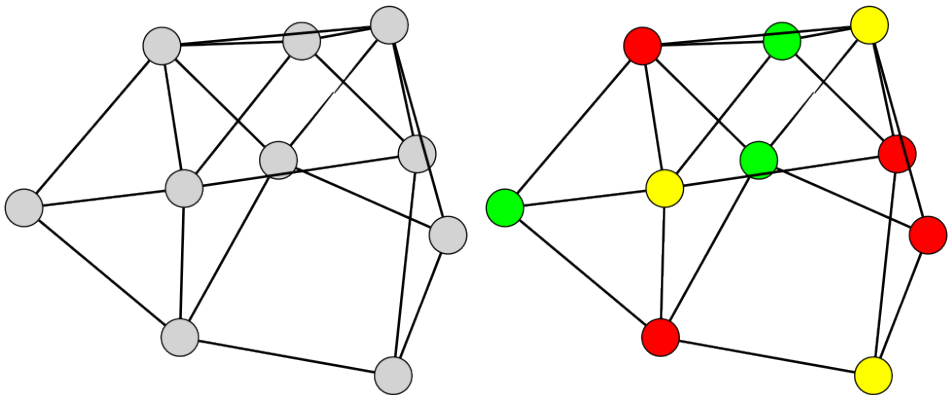

Figure 1: A graph $G = (V, E)$ with $|V| = 10$ vertices on the left and a feasible 3-coloring on the right.

The GCP can be posed as a decision or as an optimization problem. In the decision version, also called the (vertex) k-coloring problem, the question to be answered is whether for some given k a feasible k-coloring exists. The optimization version of the GCP asks for the smallest number k such that a feasible k-coloring exists; for a graph G, this number is called the *chromatic number* χ_G and the problem is thus called the chromatic number problem. For some graphs the chromatic number is known. A famous example is the four color theorem: if a graph is planar then it admits a feasible 4-coloring. However, deciding for a planar graph whether it also admits a 3-coloring has been shown to be a NP-complete problem. For general graphs it remains NP-complete and, consequently, the chromatic number problem is NP-hard (Karp 1972).

2 Random Graphs

Random graphs are generated by deciding a number of vertices and including an edge between any pair of vertices with probability p. They are typically indicated by G_p In this assignment you will be asked to find feasible colorings that use the fewest possible colors to random graphs with $|V| = 1000$ and $p = 0.5$.

3 Your task

Six instances of random graphs plus the graph with 10 vertices of Figure 1, useful for debugging purposes, are made available here:

http://www.imada.sdu.dk/~marco/DM811/Color-0/instances/.

The format of the instance files is the following. 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 of form:

  - `p edge n m` where n is the number of vertices (to be numbered 1..n) and m the number of edges.

- `e` Edge: is of the form `e n1 n2` where `n1` and `n2` are the endpoints of the edge.

A starting skeleton in Java and C++ for reading these instances is available at the course web page:

- http://www.imada.sdu.dk/~marco/DM811/Color-0/FrameworkJava.tgz
- http://www.imada.sdu.dk/~marco/DM811/Color-0/FrameworkC++.tgz

Feel free to add and remove data structures.
Note that the Java framework is much less tested than the C++ one. It is possible to use other programming languages (eg, python) but there is no skeleton available for those.

Your task is to design, implement and execute a solution algorithm for the chromatic number problem. The algorithm should find the best possible solution in one minute time on a desktop computer. The solution does not need to be optimal, but it will be compared against the other submitted on the basis of the number of colors used.
You must upload your program at:

http://www.imada.sdu.dk/~marco/DM811/Color-0

The web page will run your program, store the solution and provide an analysis of results for the assessment of your algorithm. You can continue to submit new results until the deadline. Note that in order to pass the Obligatory Assingments from 1 to 3 your soultions must be at least proper colorings.

Instructions for the Submissions at the Web Page

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 uncompress 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 $@
```

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.