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

August 21, 2009

Marco Chiarandini

DM811 - Heuristics for Combinatorial Optimization

Assignment Sheet 1, Fall 2009

---

1 Introduction

This is an assignment that will run throughout the whole course. It aims at letting the student experiment problem solving skills and gain practical experience on the methods learned in class. Moreover, it favors the development of a working environment that can be reused in the final exam project.

The experience gained and the analysis of results attained will be used for class discussions. Students will have to publish computational results in the format required in their journal blog. These results will be used by the lecturer for examples of analysis.[1]

Students are suggested to work in pairs above all in the part that concerns coding.

An update of tasks is expected with weekly cadence. At the beginning of the course the tasks aim at letting the student design its own solutions without external bias. As the course proceeds the tasks will become more specific and aimed at an active application of the knowledge acquired.

There is no evaluation to the assignment, and its accomplishment is not compulsory. However, the assignment closely reflects the content of the final project and students are highly recommended to use the assignment as a means to prepare for the exam.

2 Introduction

The graph (vertex) coloring problem (GCP) consists in finding an assignment of colors to vertices of a graph in such a way that no adjacent vertices receive the same color. 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 $\phi : V \mapsto \Gamma$, 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*.

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.

[1] Hence in order to make computational times comparable it is good that you run your experiments in the IMADA machines. This is also a recommendation for the final project as it makes sure that your programs will run on IMADA machine, which are those where your programs will be checked.

0	1	2	3	4
3	4	0	1	2
1	2	3	4	0
4	0	1	2	3
2	3	4	0	1

0	1	2	3	4	5	6
5	6	0	1	2	3	4
3	4	5	6	0	1	2
1	2	3	4	5	6	0
6	0	1	2	3	4	5
4	5	6	0	1	2	3
2	3	4	5	6	0	1

Figure 1: A solution to the 5×5 queen graph and to the 7×7 queen graph.

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 an NP-complete problem. For general graphs it remains NP-complete and consequently the chromatic number problem NP-hard (Karp 1972).

3 Coloring Queen Graphs

A famous chess puzzle asks to dispose n queens on an $n \times n$ chessboard such that none of them is able to attack any other using the standard chess queen's moves. Thus, a solution requires that no two queens share the same row, column, or diagonal.

Derived from this puzzle, the $n \times n$ queen graph has the squares of $n \times n$ chessboard for its vertices and two such vertices are adjacent if, and only if, queens placed on the two squares attack each other. For the $n \times n$ queen graph to be n colorable corresponds to dispose in a $n \times n$ chessboard $n \times n$ queens subdivided into n subgroups such that no two queens belonging to the same group can attack each other. See Figure 1 for an example, where colors are represented by the integers $\{1, \ldots, n\}$ and each integer indicates a queen group.

The $n \times n$ queen graph is n-colorable whenever $n \bmod 6 \equiv 1$ or 5. This condition is sufficient for n-colorability of the $n \times n$ queen graph. It is also necessary when $n < 12$. Thus none of the $n \times n$ queen graphs with $n = 2, 3, 4, 6, 8, 9$ is n-colorable, while for $n = 5$ and $n = 7$ a feasible solution is reported in Figure 1. For $n \geq 12$ counterexamples have been found, eg, for $n = 12, 14, 15, 16, 18, 20, 21, 22, 24, 28, 32$, that are n-colourable in spite of the fact that the condition does not hold.

4 Your tasks

1. Write a program that outputs the queen graphs in the DIMACS format.

   This format consists of a file in which 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.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
col.	3	4	5	5	7	7	9	10	11	12	12	–	14	15	16	
n	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
col.	18	–	20	21	22	–	24	–	–	–	28	–	–	–	32	

Table 1: Known solutions to the $n \times n$ queen graphs.

2. Implement a fast solver for the queen graphs and report the largest of the graphs in Table 1 for which you could find a coloring of the given number. For the graphs with a dash it is not known whether an n-coloring exists.

   Check the correctness of the coloring found with the program made available in the Assignments page. The program takes in input the problem instance file in the format described above and a solution file. The solution file reports a column of numbers corresponding to the colors assigned to each vertex. After each entry the character `\n` (new line) has to be printed. Colors start from 1 and the first color in the column represents the color assigned to vertex 1. All vertices must receive a color, if your solution has uncolored vertices use a dummy color for them.

Appendix

These are a few practical issues that it is worth knowing in preparation of the exam.

Delivery

The project will be handed in electronically. The program implementation must be handed in within the given deadline. This is done by submitting through the Blackboad system an archive thus organized:

Main directory:

```
CPRN/
```

where CPRN is the student's CPR number (eg, 030907-4089) and content:

```
CPRN/README
CPRN/doc/
CPRN/src/
```

The directory `doc` may contain the written report, that will be the main source for the evaluation of the project. The report has to be kept anonymous. The file README reports the name of the student and the manual for the compilation of the program. The directory `src` contains the source files, which may be in C, C++, Java or python. If needed a Makefile can be included either in the root directory or in `src`. After compilation the executable must be placed in `src`. For java programs, a jar package can also be submitted.

Programs must work on IMADA's computers under Linux Ubuntu operating system and with the compilers and other applications present on IMADA's computers. You are free to develop your program at home, but it is your own responsibility to transfer the program to IMADA's system and make the necessary adjustments such that it works at IMADA.[2]

Program Options and Output

The executable must be called `gcp`. It will be run by typing in the directory `CPRN/src/`:

```
gcp -i INSTANCE -t TIME -s SEED -o OUTPUT
```

- -i INSTANCE to load the data associated with the file INSTANCE.
- -t TIME to stop the program execution after TIME seconds. The test machine could not be totally dedicated at the moment of execution.
- -s SEED to initialize the random generator.
- -o OUTPUT the file name where the solution is written

[2]Past issues: 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.

For example: `gcp -i queen8_8.col -o queen8_8.sol -t 300 -s 1` will run the program on the instance `queen8_8.col` opportunely retrieved from the given path for 300 seconds with random seed 1 and write the solution in the file `queen8_8.sol`. It is advisable to have a log of algorithm activities during the run. This can be achieved by printing on the standard error or in a file (which maybe determined with a option -o filename) further information. A suggested format is to output a line whenever a new best solution is found containing at least the following pieces of information:

```
best 853 col 10 time 0.000000 iter 0 par_iter 0
```

The correctness of a solution will be checked.