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

October 1, 2013

Marco Chiarandini

DM811 - Heuristics for Combinatorial Optimization

Assignment 2, Autumn 2013

Submission deadline: Monday, September 30, 2013 at 12:00.

Read all this document before starting to work.

- The task of this assignment is to develop a stochastic local search heuristic for solving the chromatic number problem as defined in Assignment 0.

  In the design of the local search you will have to deal with the two aspects of the problem: optimization, i.e., reducing the number of colors and feasibility, i.e., two adjacent vertices must be colored differently.

  There are at least three ways to approach these aspects in local search:

  - solving a sequence of feasibility problems
  - staying in the space of feasible solutions
  - considering both feasible and infeasible solutions during the search.

  The following are alternative design choices that you may consider in the definition of the solution representation, the neighborhood function and the evaluation function. Some choices may be more promising than others.

  - number of colors in local search: {fixed, free}
  - assignment of colors to V: {complete, partial}
  - type of coloring: {proper, improper}

  The programs implementing the algorithm that you design will be tested on uniform random graphs of 1000 vertices and of different edge densities, like those from Assignment 0. The programs will be given 60 seconds of computation time on each instance.

- As in Assignment 1, you have to hand in a functioning program that will take from command line the input instance, an output file for the solution and a random seed to reproduce the run of the algorithm. The format of input files and solutions remains the same as for Assignment 0 and 1.

  In order to pass the assignment, your program must return valid results to all test instances and hence appear in the page of the analysis of results.

- You must include a short report of maximum three pages in the directory `doc/` of your submission. In the document you describe the final local search algorithm implemented and the experimental analysis conducted. Keep the document anonymous.

- Differently from the previous assignments you are asked to conduct some preliminary experimental analysis locally before submitting the best algorithm you discovered.

  You can submit only once.

  When you submit, your algorithm will be run, if the runs succeed the results will be saved and your submission closed, if some error occurred you will receive a notification and you will be allowed to resubmit. Results will be shown only after the deadline expired.

- Make sure that the program takes in input from the command line a integer parameter specifying the time limit *in seconds*. The flag for identifying this parameter is `-t` (see also below). You should therefore make sure that your program writes the solution in the output file before terminating. If not terminating within the time limit your program will be killed and if no solution is found you will have an error in your submission.

  Do not include the instance files in your submission. The maximum size of your archive is 2MB.

- In the standard output when you program finishes write a line:

  ```
  #iterations [number]
  ```

  where `[number]` is the number "moves", that is, the number of solutions visited during the search.

- An important test that you may make is to compare the effect of the initial solution on your local search. For example, you can test two choices for this component of the local search: the coloring returned by your construction heuristic from the previous assignment against a completely random initial coloring. Report the experimental analysis in your document in `doc/`.

Submission instructions

The part below is copied from Assignment 0, nothing has changed. The description of the algorithm 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 $@
```

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.