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

October 20, 2011
Marco Chiarandini

DM811 - Heuristics for Combinatorial Optimization

Exam Project, Fall 2011

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 serve 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 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 updates on the web page.

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

Friday, November 4, 2011 at 8:00 AM.

The submission procedure is the following:

- choose the course DM811 in BlackBoard,
- choose "Exam Project Hand In" 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.

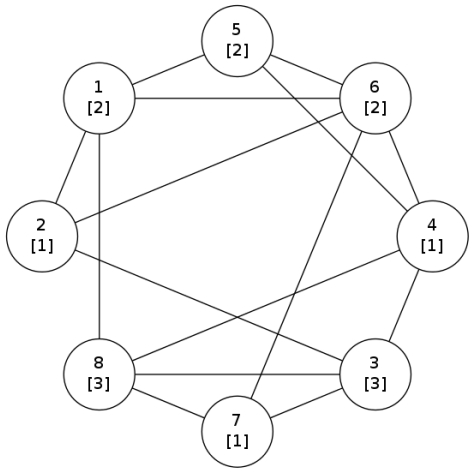
Remark 5 Make sure you have read the whole document before you start to work.

---

1 The Interval Graph Coloring Problem

Several variations and generalizations of the classical graph coloring problem arise when modeling and solving real-life problems. For example, the number of colors assigned to a vertex can be more than one, and conditions can be imposed on the colors assigned to the vertices.

Given a graph $G = (V, E)$ and strictly positive integer weights $w : V \to \mathbf{N}$, a *k-interval coloring* of G is a function I that assigns an interval $I(u) \subseteq \{1, ..., k\}$ of consecutive integers (called colors) to each vertex $u \in V$ such that:




```
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1[2]:     * *
2[1]:           *
3[3]: * * *
4[1]:       *
5[2]:           * *
6[2]: * *
7[1]:       *
8[3]:         * * *
```

Figure 1: On the left a graph with 8 vertices labeled by numbers from 1 to 8. The number within square bracket is the weight of the vertex. On the right an interval coloring. Each vertex receives a number of consecutive numbers equal to its weight and the intervals of adjacent vertices do not overlap. The solution uses 7 colors and the span is 6.

(i) for every vertex $u \in V$, $|I(u)| = w(u)$ and

(ii) for every pair of adjacent vertices u and v, $I(u) \cap I(v) = \emptyset$.

For a fixed integer k, the k-interval graph coloring problem (k-ICP) asks to find a k-interval coloring of G. The interval coloring problem (ICP) asks to determine the smallest integer k, called *interval chromatic number* of G and denoted $\chi_{\text{int}}(G)$, such that there exists a k-interval coloring of G.

For illustration, Figure 1 on the right shows a 7-interval coloring for the graph G depicted on the left, where the numbers between brackets correspond to weights on vertices.

The classical vertex coloring problem is a special case of the ICP and k-ICP where $w(u) = 1$ for all vertices $u \in V$. Stockmeyer showed in 1976 that the interval-coloring problem is NP-complete, even when restricted to interval graphs and vertex weights in $\{1, 2\}$ (see problem SR2 in [GJ79]).

The ICP models the problem of memory allocation in compilers [Fab79]. In order to reduce the total memory consumption of source-code objects (simple variables, arrays, and structures), the compiler can make use of the fact that the memory regions of two objects are allowed to overlap, provided that the objects do not "interfere" at run-time. This problem can be abstracted as the interval-coloring problem, as follows. The source-code objects correspond to vertices of our graph, run-time interference between pairs of source code objects is represented by edges of the graph, the amount of memory needed for each source-code object is represented by the weight of the corresponding vertex, and the assignment of memory regions to source code objects is represented by the assignment of intervals to vertices of the graph. Minimizing the size of the union of intervals corresponds to minimizing the amount of memory-allocation.

The ICP arises also in course timetabling problems where lectures of different length must be scheduled in consecutive time slots [CE83, CS91]. In this case, lectures are the vertices of a graph and edges connect courses or exams that cannot be scheduled at the same time because they share students or teachers. Here, time slots are the colors and weights indicate the amount of time slots to be allocated for each lecture.

2 Further definitions and known properties

This is a list of definitions and known properties that may be used in the design of solution algorithms.

1. A k-interval coloring is said *improper* (or illegal) if two adjacent vertices u and v have common colors, i.e. $I(u) \cap I(v) \neq \emptyset$ for an edge $uv \in E$. In this case, the edge uv is said to be *conflicting*.

2. The *total vertex chromaticity* $w(G)$ defined as

$$w(G) = \sum_{u \in V} w(u)$$

   is an upper bound to the interval chromatic number χ_{int}.

3. If G is a bipartite graph, we have $\chi_{\text{int}}(G) = \max\{w(u) + w(v) \mid uv \in E\}$. If G is a clique then, $\chi_{\text{int}}(G)$ is equal to $\sum_{u \in V} w(u)$. For an arbitrary graph G, any maximal complete induced subgraph K has $\chi_{\text{int}}(K) \leq \chi_{\text{int}}(G)$, hence $\chi_{\text{int}}(K)$ is a lower bound to $\chi_{\text{int}}(G)$. The best lower bound determined in this way, is given by $\max\{\chi_{\text{int}}(K) \mid K \text{ is a complete subgraph of } G\}$.

   In Figure 1 the best lower bound is 7, given by the clique $\{3, 4, 8\}$. Since the coloring presented as solution uses 7 colors, the solution shown is optimal and 7 is the interval chromatic number for that graph.

4. Let $A_V(u)$ be the set of vertices that are adjacent to u in the subgraph of G induced by V and let the vertex u be itself included in $A_V(u)$. The *vertex chromatic degree* of a vertex u, denoted by $\delta_\chi(u)$, is the maximum number of different colors that may be needed to color the vertices in $A_V(u)$, that is,

$$\delta_\chi(u) = \sum_{v \in A_V(u)} w(v).$$

5. The *span* of a coloring I is defined as $span(I) = \max_{u \in V} I(u) - \min_{u \in V} I(u) + 1$. The number of colors used in a k-interval coloring corresponds to the span, i.e. $k = |\bigcup_{v \in V} I(v)| = span(I)$. Thus, minimizing k for $\Gamma = \{1..k\}$ is equivalent to minimizing the span.

6. A coloring of a vertex-weighted graph $G = (V, E)$ determined by the partitioning of the vertices into color classes $C_1, C_2, \ldots, C_k$ *induces* an interval-coloring of G, that can be constructed as follows. For each i, $1 \leq i \leq k$, let $v_i \in C_i$ be the vertex with maximum weight in C_i. Let $H(1) = 0$, and for each i, $2 \leq i \leq k$, let $H(i) = \sum_{j=1}^{i-1} w(v_j)$. For each vertex $v \in C_i$, we set $I(v) = \{H(i)..H(i) + w(v)\}$. Clearly, no two vertices in distinct color classes have overlapping intervals and, therefore, this is a valid interval-coloring of G. The span of this interval-coloring is $\sum_{i=1}^{k} w(v_i)$.

7. Another generalization of the classical vertex coloring problem is the BANDWIDTH COLORING PROBLEM (BCP).[1] Given a graph G and strictly positive integer weights $\delta : V \to E$, a *bandwidth coloring* of G is a function c that assigns an integer (called a color) to each vertex $u \in V$ so that $|c(i) - c(j)| \geq \delta_{uv}$ for all edges $uv \in E$. The BCP asks to determine a bandwidth coloring with minimum span. The graph coloring problem is a special case of the bandwidth coloring problem with $\delta_{uv} = 1$ for all edges $uv \in E$.

8. In [BČH10] it is shown that an optimal solution of the ICP can be obtained by solving a series of BCPs. The proposed algorithm is presented in Figure 2. It consists in first transforming the given weighted graph $G = (V, E, w)$ in a graph G' with even weights, i.e. $G' = (V, E, 2w)$; then, determining optimal bandwidth colorings c_{Δ_k} in graphs G_{Δ_k}, defined below, for various values of k. Finally, transforming the bandwidth coloring for the graph G_{Δ_k} to an interval coloring for the graph G' as indicated in Line 10 of EVENREDUCTION and the interval coloring for graph G' to one for G as indicated in Line 6 of GENERALREDUCTION.

[1] BCP is a special case of the T-coloring problem and it is also known as distance coloring problem

1 **function** GENERALREDUCTION;
2 **input** A graph $G = (V, E)$ with weights $w(u)$ on the vertices $u \in V$;
3 **output** An optimal interval coloring I of G;
4 Construct G' from G by multiplying every weight $w(u)$ by 2;
5 Determine a compact optimal interval coloring I' of G' using EVENREDUCTION;
6 Set $I(i) = \{\lceil \frac{\min I'(i)}{2} \rceil, \ldots, \lceil \frac{\min I'(i)}{2} \rceil + w(u) - 1\}$ for all vertices $u \in V$.

1 **function** EVENREDUCTION;
2 **input** A graph $G = (V, E)$ with weights $w(u)$ on the vertices $u \in V$;
3 **output** An optimal interval coloring I of G;
4 Set $k := 1$ and $\Delta_1 := 0$;
5 Determine an optimal bandwidth coloring c_{Δ_k} of G_{Δ_k};
6 Set $\Delta_{k+1} := span(c_{\Delta_k}) - 1$;
7 **if** $|c_{\Delta_k}(\beta) - c_{\Delta_k}(\alpha)| < \Delta_{k+1}$ **then**
8 set $k := k + 1$ and goto line 5;
9 **else**
10 set $I(i) = \{c_{\Delta_k}(u) - \frac{w(u)}{2}, \ldots, c_{\Delta_k}(u) + \frac{w(v)}{2} - 1\}$ for all $u \in V$.

Figure 2: The algorithm for obtaining optimal interval coloring by solving series of bandwidth coloring problems

For a graph $G = (V, E)$ with even weights $w(u)$ on the vertices $u \in V$ and any positive integer Δ the edge-weighted graph G_Δ input to the BCP is constructed as follows. Include all vertices V and edges E and add two vertices α and β linked to all other vertices in V. Set the weights:

- $\delta_{uv} = \frac{w(u)+w(v)}{2}$ for all $uv \in E$
- $\delta_{\alpha u} = \delta_{\beta u} = \frac{w(u)}{2}$ for all $u \in V$
- $\delta_{\alpha\beta} = \Delta$

The number h^* indicates the number of bandwidth problems for G' that have to be solved. To reduce this number of iterations, instead of starting from $\Delta_1 = 0$, one could start from a lower bound to the interval chromatic number, for example the one provided by a clique of the graph G multiplied by 2. With heuristics the increasing procedure is not appropriate because they cannot provide a proof of non-existence in Line 5 of EVENREDUCTION. One could however start from an upper bound and decrease Δ_k until no solution to the BCP can be found.

9. An Integer Programming model for the ICP is the following. Let x_{uk} be a binary variable that is 1 if the color interval of the vertex u starts at k, i.e. $\min I(u) = k$, and zero otherwise; and let y_k be an auxiliary variable indicating whether the color k is used. Further, let $\mathcal{C}$ be a collection

60	120	240
T-G.5.5-60.0.5.1.col	T-G.5.5-120.0.5.1.col	T-G.5.5-240.0.5.1.col
T-G.5.5-60.0.5.2.col	T-G.5.5-120.0.5.2.col	T-G.5.5-240.0.5.2.col
T-G.5.5-60.0.5.3.col	T-G.5.5-120.0.5.3.col	T-G.5.5-240.0.5.3.col
T-G.5.5-60.0.5.4.col	T-G.5.5-120.0.5.4.col	T-G.5.5-240.0.5.4.col
T-G.5.5-60.0.5.5.col	T-G.5.5-120.0.5.5.col	T-G.5.5-240.0.5.5.col
T-G.5.5-60.0.5.6.col	T-G.5.5-120.0.5.6.col	T-G.5.5-240.0.5.6.col
T-G.5.5-60.0.5.7.col	T-G.5.5-120.0.5.7.col	T-G.5.5-240.0.5.7.col
T-G.5.5-60.0.5.8.col	T-G.5.5-120.0.5.8.col	T-G.5.5-240.0.5.8.col
T-G.5.5-60.0.5.9.col	T-G.5.5-120.0.5.9.col	T-G.5.5-240.0.5.9.col
T-G.5.5-60.0.5.10.col	T-G.5.5-120.0.5.10.col	T-G.5.5-240.0.5.10.col

Table 1: The set of test instances grouped into different vertex set size.

of cliques of G such that each edge $uv \in E$ belongs to some clique C of $\mathcal{C}$ (for example $\mathcal{C} = E$).

$$\min \; z \tag{1}$$

$$\sum_{k \in \Gamma} x_{uk} = 1 \qquad \forall u \in V \tag{2}$$

$$\sum_{u \in C} \sum_{l=\max\{0,k-w(u)\}}^{k} x_{ul} \leq y_k \qquad \forall C \in \mathcal{C}, k \in \Gamma \tag{3}$$

$$k y_k \leq z \qquad \forall k \in \Gamma \tag{4}$$

$$x_{uk} \in \{0,1\} \qquad \forall u \in V, k \in \Gamma \tag{5}$$

$$y_k \in \{0,1\} \qquad \forall k \in \Gamma \tag{6}$$

The set of colors Γ is made of all consecutive integers from 1 to an upper bound. Constraints (2) ensure that each vertex has assigned an interval of colors by imposing that the interval starts at some color. Constraints (3) ensure that the intervals assigned to vertices that belong to a clique do not overlap. Constraints (4) together with the objective (1) minimize the largest color used.

Using a MIP solver like `gurobi` this model yields very fast solutions to instances of 30 vertices but is unable to find a feasible solution after one hour on the instances that are the object of this project (see next section).

3 Project Requirements

The aim of the project is to study heuristic algorithms for finding the interval chromatic number of arbitrary random graphs and compare the heuristics on the test instances of Table 1.

The name of the instances has the following format: `T-G.w.5-SIZE.DENSITY.SEED.col`, where `w` is the maximum weight on the vertices, `SIZE` is the number of vertices, `DENSITY` is the edge density and `SEED` is the seed number for the generation of the instance. The instances can be downloaded from the course web page. The instance `clementson.col` used in Figure 1 is also included for testing purposes.

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

1. Design and implement two or more construction heuristics and show that they perform better than the solution obtained by passing a random permutation of vertices to a procedure that given a set of vertices already colored assigns the smallest (i.e, the first) feasible interval to the new vertex to color.

2. Design and implement one or more local search algorithms.
3. Design and implement an effective algorithm enhancing the heuristics developed at one or both of the previous two points using stochastic local search methods and metaheuristics.
4. For all the methods above carry out an experimental analysis and draw sound conclusions.

In the experimental assessment of any algorithm limit the maximum computation to one minute.[2]

4 Remarks

Remark 1 For each point above a description must be provided in the report of the work undertaken. In particular for the best algorithms 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).

Remark 2 The results of the experiments must be reported either in graphical form or in form of tables or both. Moreover, for the best solver resulting from the point 4, a table must be provided with the best results for each specific instance of Table 1.

Remark 3 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 4 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 5 In the project requirements of Sec. 3 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!

[2] Times refer to machines in IMADA terminal room.

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 in each line the set of colors assigned to the corresponding vertex.

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

```
$> cat clementson.sln
3 4
6
1 2 3
4
6 7
1 2
4
5 6 7
```

Appendix C Solution Validator

A program to check the validity of a given solution is 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 clementson.col -s clementson.sln -p 6
```

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. In both submissions *do not put your name in the author field of the report, instead put your CPR number.* The official receipt will be obtained at the BlackBoard submission.

This section describes how you must organize your electronic submission.

Main directory:

```
CPRN/
```

where CPRN is the student's *CPR number without the last four digits* (eg, 030907) and content:

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

The directory `doc` contains a pdf or postscript 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. The directory `data` contains the instances.

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.[3]

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

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

where the flags indicate:

- `-i INSTANCE` the input instance;
- `-t TIME` the time limit in seconds;
- `-s SEED` the random seed;
- `-o OUTPUT` the file name where the solution is written.

For example:

```
gcp -i data/clementson.col -o clementson.sln -t 60 -s 1 > clementson.log
```

will run the program on the instance `clementson.col` opportunely retrieved from the given path for 180 seconds with random seed 1 and write the solution in the file `clementson.sln`.

In its default mode, the program must run the best algorithm developed and must print on the standard output **only one single number** at the end of the run corresponding to the quality of the best interval coloring found during the run.

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`. Process times include the time to read the instance.

[3] 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`.

References

[BČH10] M. Bouchard, M. Čangalović, and A. Hertz. On a reduction of the interval coloring problem to a series of bandwidth coloring problems. *Journal of Scheduling*, 13:583–595, 2010.

[CE83] A. T. Clementson and C. H. Elphick. Approximate colouring algorithms for composite graphs. *The Journal of the Operational Research Society*, 34(6):pp. 503–509, 1983.

[CS91] Mirjana Cangalovic and Jan A.M. Schreuder. Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths. *European Journal of Operational Research*, 51(2):248–258, 1991.

[Fab79] J. Fabri. Automatic storage optimization. In *ACM SIGPLAN Notices: Proceedings of the ACM SIGPLAN on Compiler Construction 14*, volume 8, pages 83–91, 1979.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, USA, 1979.