Appendix A
Computing Resources

A.1 Algorithm User Guide

This section contains instructions on how to compile and use the implementations of the algorithms described in Chapters 2 and 4 of this book. These can be downloaded directly from:

`http://rhydlewis.eu/resources/gCol.zip`

Once downloaded and unzipped, we see that the directory contains a number of subdirectories. Each algorithm is contained within its own subdirectory. Specifically, these are:

- `AntCol` The Ant Colony Optimisation-based algorithm for graph colouring (see Section 4.1.4).
- `BacktrackingDSatur` The Backtracking algorithm based on the DSatur heuristic (see Section 4.1.6).
- `DSatur` The DSATUR algorithm (see Section 2.3).
- `HillClimber` The hill-climbing algorithm (see Section 4.1.5).
- `HybridEA` The hybrid evolutionary algorithm (see Section 4.1.3).
- `PartialColAndTabuCol` The PARTIALCOL and TABUCOL algorithms (see Sections 4.1.1 and 4.1.2 respectively).
- `RLF` The recursive largest first (RLF) algorithm (see Section 2.4).
- `SimpleGreedy` The GREEDY algorithm, using a random permutation of the vertices (see Section 2.1).

All of these algorithms are programmed in C++. They have been successfully compiled in Windows using Microsoft Visual Studio 2010 and in Linux using the GNU compiler `g++`. Instructions on how to do this now follow.

© Springer International Publishing Switzerland 2016

R.M.R. Lewis, *A Guide to Graph Colouring*,
DOI 10.1007/978-3-319-25730-3

A.1.1 Compilation in Microsoft Visual Studio

To compile and execute using Microsoft Visual Studio the following steps can be taken:

1. Open Visual Studio and click **File**, then **New**, and then **Project from Existing Code**.
2. In the dialogue box, select **Visual C++** and click **Next**.
3. Select one of the subdirectories above, give the project a name, and click **Next**.
4. Finally, select **Console Application Project** for the project type, and then click **Finish**.

The source code for the chosen algorithm can then be viewed and executed from the Visual Studio application. Release mode should be used during compilation to make the programs execute at maximum speed.

A.1.2 Compilation with g++

To compile the source code using `g++`, at the command line navigate to each subdirectory in turn and use the following command:

```
g++ *.cpp -O3 -o myProgram
```

By default this will create a new executable program called `myProgram` that can then be run from the command line (you should choose your own name here). The optimisation option `-O3` ensures that the algorithms execute at maximum speed. Makefiles are also provided for compiling all algorithms in one go, if preferred.

A.1.3 Usage

Once generated, the executable files (one per subdirectory) can be run from the command line. If the programs are called with no arguments, useful usage information is printed to the screen. For example, suppose we are using the executable file `hillClimber`. Running this program with no arguments from the command line gives the following output:

```
Hill Climbing Algorithm for Graph Colouring

USAGE:
<InputFile>  (Required. File must be in DIMACS format)
-s <int>     (Stopping criteria expressed as number of constraint
              checks. Can be anything up to 9x10^18.
              DEFAULT = 100,000,000.)
-I <int>     (Number of iterations of local search per cycle.
              DEFAULT = 1000)
```

```
-r <int>     (Random seed. DEFAULT = 1)
-T <int>     (Target number of colours. Algorithm halts if this
              is reached. DEFAULT = 1.)
-v           (Verbosity. If present, output is sent to screen.
              If -v is repeated, more output is given.)
****
```

The input file should contain the graph colouring problem to be solved. This is the only mandatory argument. This must be in the DIMACS format, as described here:

`mat.gsia.cmu.edu/COLOR/general/ccformat.ps`

For reference, an example input file called `graph.txt` is provided in each subdirectory.

The remaining arguments for each of the programs are optional and are allocated default values if left unspecified. Here are some example commands using the `hillClimber` executable:

```
hillClimber graph.txt
```

This will execute the algorithm on the problem given in the file `graph.txt`, using the default of 1,000 iterations of local search per cycle and a random seed of 1. The algorithm will halt when 100,000,000 constraint checks have been performed. No output will be written to the screen.

Another example command is:

```
hillClimber graph.txt -r 6 -T 50 -v -s 500000000000
```

This run will be similar to the previous one, but will use the random seed 6 and will halt either when 500,000,000,000 constraint checks have been performed, or when a feasible solution using 50 or fewer colours has been found. The presence of `-v` means that output will be written to the screen. Including `-v` more than once will increase the amount of output.

The arguments `-r` and `-v` are used with all of the algorithms supplied here. Similarly, `-T` and `-s` are used with all algorithms except for the single-parse constructive algorithms DSATUR, RLF and GREEDY. Descriptions of arguments particular to just one algorithm are found by typing the name of the program with no arguments, as described above. Interpretations of the run-time parameters for the various algorithms can be found by consulting the algorithm descriptions in this book.

A.1.4 Output

When a run of any of the programs is completed, three files are created:

- `cEffort.txt` (computational effort),

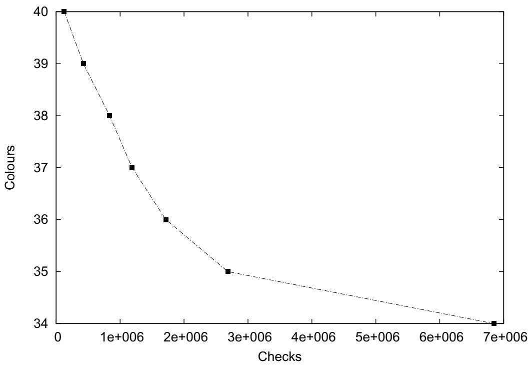
- `tEffort.txt` (time effort), and
- `solution.txt`.

The first two specify how long (in terms of constraint checks and milliseconds respectively) solutions with certain numbers of colours took to produce during the last run. For example, we might get the following computational effort file:

```
40   126186
39   427143
38   835996
37   1187086
36   1714932
35   2685661
34   6849302
33   X
```

This file is interpreted as follows: The first feasible solution observed used 40 colours, and this took 126,186 constraint checks to achieve. A solution with 39 colours was then found after 427,143 constraint checks, and so on. To find a solution using 34 colours, a total of 6,849,302 constraint checks was required. Once a row with an `X` is encountered, this indicates that no further improvements were made: that is, no solution using fewer colours than that indicated in the previous row was achieved. Therefore, in this example, the best solution found used 34 colours. For consistency, the `X` is always present in a file, even if a specified target has been met.

The file `tEffort.txt` is interpreted in the same way as `cEffort.txt`, with the right hand column giving the time (in milliseconds) as opposed to the number of constraint checks. Both of these files are useful for analysing algorithm speed and performance. For example, the computational effort file above can be used to generate the following plot:



Finally, the file `solution.txt` contains the best feasible solution (i.e., the solution with fewest colours) that was achieved during the run. The first line of this file gives the number of vertices n, and the remaining n lines then state the colour of each vertex, using labels $0, 1, 2, \ldots$.

For example, the following solution file

```
5
0
2
1
0
1
```

is interpreted as follows: There are five vertices; the first and forth vertices are assigned to colour 0, the third and fifth vertices are assigned to colour 1, and the second vertex is assigned to colour 2.

A.2 Graph Colouring in Sage

Sage is specialised software that allows the exploration of many aspects of mathematics, including combinatorics, graph theory, algebra, calculus and number theory. It is both free to use and open source. To use Sage, commands can be typed into a notebook. Blocks of commands are then executed by typing Shift + Enter next to these commands, with output (if applicable) then being written back to the notebook.

Sage contains a whole host of elementary and specialised mathematical functions that are documented online at www.sagemath.org/doc/reference/. Of particular interest to us here is the functionality surrounding graph colouring and graph visualisations. A full description of the graph colouring library for Sage can be found at:

```
www.sagemath.org/doc/reference/graphs/sage/graphs/
  graph_coloring.html
```

The following text now shows some example commands from this library, together with the output that Sage produces. In our case, these commands have been typed into notebooks provided by the online tool at SageMathCloud. This tool allows the editing and execution of Sage notebooks through a web browser and can be freely accessed online at:

```
https://cloud.sagemath.com
```

The following pieces of code each represent an individual block of executable Sage commands. Any output produced by these commands are preceded by the ">>" symbol in the following text.

To begin, it is first necessary to specify the names of the libraries we intend to use in our Sage program. We therefore type:

```
from sage.graphs.graph_coloring import chromatic_number
from sage.graphs.graph_coloring import vertex_coloring
from sage.graphs.graph_coloring import number_of_n_colorings
from sage.graphs.graph_coloring import edge_coloring
```

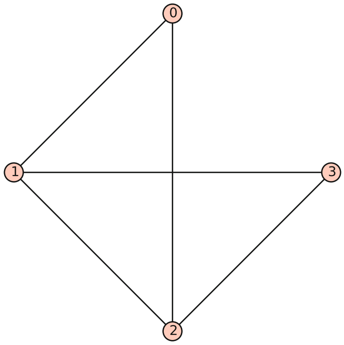
which will allow us to access the various graph colouring functions used below.

We will now generate in Sage a small graph called `G`. In our case this graph has $n = 4$ vertices and $m = 5$ edges and is defined by the adjacency matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

The first Sage command below defines this matrix. The next command then transfers this information into a graph called `G`. Finally `G.show()` draws this graph to the screen.

```
A = matrix([[0,1,1,0],[1,0,1,1],[1,1,0,1],[0,1,1,0]])
G = Graph(A)
G.show()
>>
```



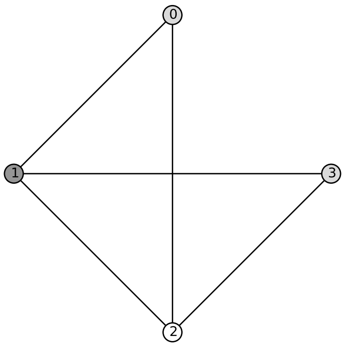
Note that, by default, Sage labels the vertices from $0,\ldots,n-1$ in this diagram as opposed to using indices $1,\ldots,n$.

We will now produce an optimal colouring of this graph. The algorithms that Sage uses to obtain these solutions are based on integer programming techniques (see Section 3.1.2). These are able to produce provably optimal solutions for small graphs such as this example; however, for larger graphs they are unlikely to return solutions in reasonable time. A colouring is produced via the following command (note the spelling of "coloring" as opposed to "colouring"):

```
vertex_coloring(G)
>> [[2], [1], [3, 0]]
```

The output produced by Sage tells us that `G` can be optimally coloured using three colours, with vertices 0 and 3 receiving the same colour. The solution returned by Sage is expressed as a partition of the vertices, which can be used to produce a visualisation of the colouring as follows:

```
S = vertex_coloring(G)
G.show(partition=S)
>>
```



Here, the partition produced by `vertex_coloring(G)` is assigned to the variable `S`, which is then used as an additional argument in the `G.show` command to produce the above visualisation.

We can also use the `vertex_coloring` function to test if a graph is k-colourable. For example, to test whether `G` is 2-colourable, we get

```
vertex_coloring(G,2)
>> False
```

which tells us that a 2-colouring is not possible for this graph. On the other hand, if we seek to confirm whether `G` is 4-colourable, we get

```
vertex_coloring(G,4)
>> [[3, 0], [2], [1], []]
```

which tells us that one way of 4-colouring the graph `G` is to not use the fourth colour!

In addition to the above, commands are also available in Sage for determining the chromatic number

```
chromatic_number(G)
>> 3
```

and for calculating the number of different k-colourings. For example, with $k = 2$ we get

```
number_of_n_colorings(G,2)
>> 0
```

which is what we would expect since no 2-colouring of `G` exists. On the other hand, for $k = 3$ we get

```
number_of_n_colorings(G,3)
>> 6
```

telling us that there are six different ways of feasibly assigning three colours to `G` (readers are invited to confirm the correctness of this result themselves, by hand or otherwise).

In addition to vertex colouring, Sage also provides commands for calculating edge colourings of a graph (see Section 5.2). For example, continuing our use of the graph `G` from above, we can use the `edge_coloring()` command to get

```
edge_coloring(G)
>> [[(0, 1), (2, 3)], [(0, 2), (1, 3)], [(1, 2)]]
```

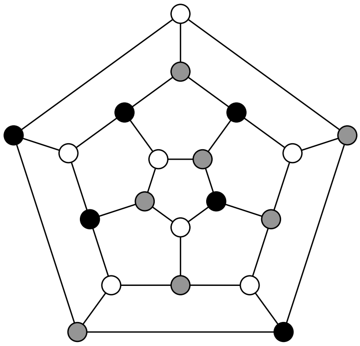
This tells us that the chromatic index of `G` is 4, with edges $\{0,1\}$ and $\{2,3\}$ being assigned to one colour, $\{0,2\}$ and $\{1,3\}$ being assigned to a second, and $\{1,2\}$ being assigned to a third.

Sage also contains a collection of predefined graphs. This allows us to make use of common graph topologies without having to type adjacency matrices. A full list of these graphs is provided at:

```
www.sagemath.org/doc/reference/graphs/sage/graphs/
  graph_generators.html
```

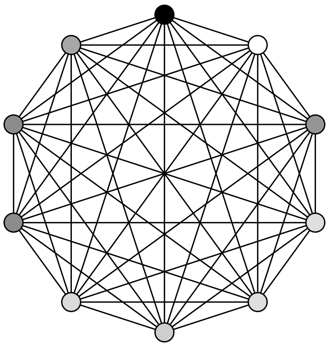
For example, here are the commands for producing an optimal colouring of a dodecahedral graph. In this case we have switched off vertex labelling to make the illustration clearer:

```
G = graphs.DodecahedralGraph()
S = vertex_coloring(G)
G.show(partition=S, vertex_labels=False)
>>
```



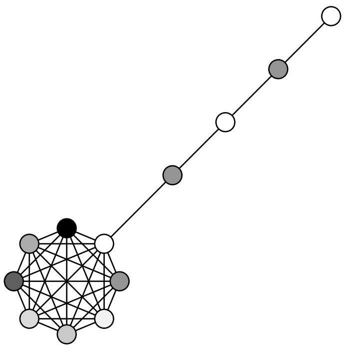
Here is an optimal colouring of the complete graph with ten vertices, K_{10}:

```
G = graphs.CompleteGraph(10)
S = vertex_coloring(G)
G.show(partition=S, vertex_labels=False)
>>
```



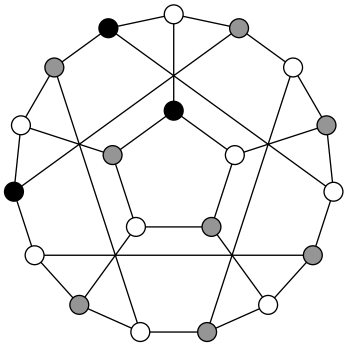
So-called lollypop graphs are defined by a path of n_1 vertices representing the "stick" and a complete graph K_{n_2} to represent the "head". Here is an example colouring using a graph with $n_1 = 4$ and $n_2 = 8$:

```
G = graphs.LollipopGraph(8,4)
S = vertex_coloring(G)
G.show(partition=S, vertex_labels=False)
>>
```



Our next graph, the "flower snark" is optimally coloured as follows:

```
G = graphs.FlowerSnark()
S = vertex_coloring(G)
G.show(partition=S, vertex_labels=False)
>>
```

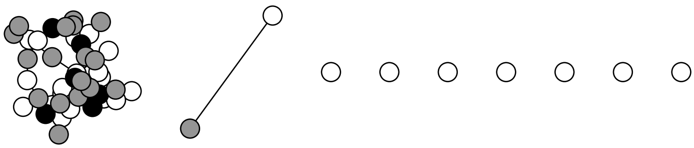


As we can see, this graph is 3-colourable. However, we can confirm that it is not planar using the following command:

```
G.is_planar()
>> False
```

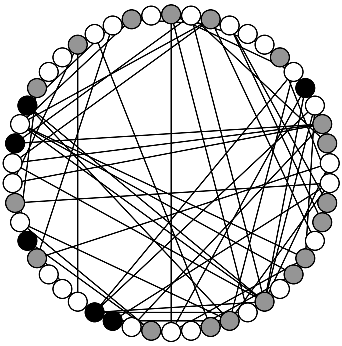
Finally, Sage also allows us to define random graphs $G_{n,p}$ that have n vertices and edge probabilities p (see Definition 2.15). Here is an example with $n = 50$ and $p = 0.05$:

```
G = graphs.RandomGNP(50,0.05)
S = vertex_coloring(G)
G.show(partition=S, vertex_labels=False)
>>
```



It can be seen that this particular graph is 3-colourable, although the default layout of this graph is not very helpful, with the connected component on the left being very tightly clustered. If desired we can change this layout so that vertices are presented in a circle:

```
G.show(vertex_labels=False, layout='circular', partition=S)
>>
```



This, arguably, gives a clearer illustration of the graph.

A.3 Graph Colouring with Commercial IP Software

The following code demonstrates how the graph colouring problem might be specified using integer programming methods and then solved using off-the-shelf optimisation software. This particular example, relating to the first IP model discussed in Section 3.1.2, is coded in the Xpress-Mosel language, which comes as part of the FICO Xpress Optimisation Suite. Comments in the code are preceded by exclamation marks.

```
model GCOL
   !Gain access to the Xpress-Optimizer solver
   uses "mmxprs";

   !Define input file
   fopen("myGraph.txt",F_INPUT)

   !Define the integers used in the program
   declarations
      n,m,v1,v2: integer
   end-declarations

   !Read the num of vertices and edges from the input file
   read(n,m)
   writeln("n = ",n,", m = ",m)

   !Declare the decision variable arrays
   declarations
      X: array(1..n,1..n) of mpvar
      Y: array(1..n) of mpvar
   end-declarations
   !And make all the variables binary
   forall (i in 1..n) do
      forall (j in 1..n) do
         X(i,j) is_binary
      end-do
      Y(i) is_binary
   end-do

   !Specify that each vertex should be assigned to exactly
   !one colour
   forall (i in 1..n) do
      sum(j in 1..n) X(i,j) = 1
   end-do

   !Now read in all of the edges and define the constraints
   write("E = {")
   forall (j in 1..m) do
      read(v1,v2)
      forall (i in 1..n) do
         X(v1,i) + X(v2,i) <= Y(i)
      end-do
      write("{",v1,",",v2,"}")
   end-do
```

```
   writeln("}")

   !Now specify the objective function
   objfn := sum(i in 1..n) Y(i)

   !Now run the model
   writeln
   writeln("Running model...")
   minimize(objfn)
   writeln("...Run ended")

   !Finally write the output to the screen
   writeln
   writeln("Cost (number of colours) = ",getobjval)
   writeln
   writeln("X = ")
   forall (i in 1..n) do
      forall (j in 1..n) do
         write(getsol(X(i,j))," ")
      end-do
      writeln
   end-do
   writeln
   writeln("Y = ")
   forall (j in 1..n) do
      write(getsol(Y(j))," ")
   end-do
   writeln
   writeln
   forall (i in 1..n) do
      write("c(v_",i,") = ")
      forall (j in 1..n) do
         if(getsol(X(i,j))=1) then
            writeln(j)
         end-if
      end-do
   end-do

end-model
```

The above program starts by reading in a graph colouring problem from a text file (called `myGraph.txt` in this case). The objective function and constraints of the problem are then specified, before the optimisation process itself is invoked using the `minimize(objfn)` command. In this case the optimisation process is terminated only once a provably optimal solution has been found. However, other stopping conditions can also be specified if needed. Finally, the solution is written to the screen in a readable way.

Here is some example input that can be read in by the above program. The first two lines give the number of vertices and edges, n and m, respectively. The m edges then follow, one per line. This particular example corresponds to the graph shown in Figure 3.2.

```
8
12
1 2
1 3
1 4
2 5
2 6
2 8
3 4
3 7
4 7
5 8
6 8
7 8
```

On completion of the program, the following output is produced:

```
n = 8, m = 12
E = {{1,2}{1,3}{1,4}{2,5}{2,6}{2,8}{3,4}{3,7}{4,7}{5,8}{6,8}
     {7,8}}

Running model...
...Run ended

Cost (number of colours) = 3

X =
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0

Y =
1 1 0 0 0 0 1 0

c(v_1) = 1
c(v_2) = 2
c(v_3) = 7
c(v_4) = 2
c(v_5) = 1
c(v_6) = 1
c(v_7) = 1
c(v_8) = 7
```

It can be seen that the cost of the solution (the number of colours being used) equals the chromatic number for this graph as expected. Note that, in this case, the colours with labels 1, 2 and 7 are being used to colour the vertices as opposed to 1, 2, and 3, which is permitted by this particular formulation.

A.4 Useful Web Links

Here are some further web resources related to graph colouring. A page of resources maintained by Joseph Culberson featuring, most notably, a collection of problem generators and C code for the algorithms presented by Culberson and Luo (1996) can be found at:

`webdocs.cs.ualberta.ca/~joe/Coloring/`

An excellent bibliography on the graph coloring problem, maintained by Marco Chiarandini and Stefano Gualandi can also be found at:

`www.imada.sdu.dk/~marco/gcp/`

A large set of graph colouring problem instances has been collected by the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) as part of their DIMACS Implementation Challenge series. These can be downloaded at:

`mat.gsia.cmu.edu/COLOR/instances.html`

These problem instances have been used in a large number of graph colouring-based papers and are written in the DIMACS graph format, a specification of which can be found in the following (postscript) document:

`mat.gsia.cmu.edu/COLOR/general/ccformat.ps`

Note that these instances can also be viewed via a text editor. A summary of these instances, including their best known bounds, is maintained by Daniel Porumbel and is available at:

`www.info.univ-angers.fr/pub/porumbel/graphs/`

The fun graph colouring game *CoLoRaTiOn*, which is suitable for both adults and children, can be downloaded from:

`http://vispo.com/software/`

The goal in this game is to achieve a feasible colouring within in a certain number of moves. The difficulty of each puzzle depends on a number of factors, including its topology, whether you can see all of the edges, the number of vertices, and the number of available colours.

Finally, C++ code for the random Sudoku problem instance generator used in Section 5.4.1 of this book can be downloaded from:

`rhydlewis.eu/resources/sudokuGeneratorMetaheuristics.zip`

A Sudoku to graph colouring problem converter can also be found at:

`rhydlewis.eu/resources/sudokuToGCol.zip`

When compiled, this program reads in a single Sudoku problem (from a text file) and converts it into the equivalent graph colouring problem in the DIMACS format mentioned above.

References

K. Aardel, S. van Hoesel, A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for the frequency assignment problems. *4OR : Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(4): 1–40, 2002.

A. Anagnostopoulos, L. Michel, P. van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

I. Anderson. Kirkman and GK_2n. *Bulletin of the Institute of Combinatorics and its Applications*, 3:111–112, 1991.

K. Appel and W. Haken. Solution of the four color map problem. *Scientific American*, 4:108121, 1977a.

K. Appel and W. Haken. Every planar map is four colorable. Part I. Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977b.

K. Appel and W. Haken. Every planar map is four colorable. Part II. Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977c.

C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operations Research*, 151:379–388, 2003.

T. Bartsch, A. Drexl, and S. Kroger. Scheduling the professional soccer leagues of Austria and Germany. *Computers and Operations Research*, 33(7):1907–1937, 2006.

L. Beineke and J. Wilson, editors. *Topics in Chromatic Graph Theory*. Encyclopedia of Mathematics and its Applications (no. 156). Cambridge University Press, 2015.

C. Berge. Les problémes de coloration en théorie des graphes. *Publ. Inst. Stat. Univ. Paris*, 9:123–160, 1960.

C. Berge. *Graphs and Hypergraphs*. North-Holland, 1970.

D. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1:7–66, 1992.

I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research*, 35:960–975, 2008.

© Springer International Publishing Switzerland 2016

R.M.R. Lewis, *A Guide to Graph Colouring*,
DOI 10.1007/978-3-319-25730-3

B. Bollobás. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

B. Bollobás. *Modern Graph Theory*. Springer, 1998.

W. Boyer, J. Myrvold. On the cutting edge: simplified $\mathcal{O}(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8:241–273, 2004.

D. Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4): 251–256, 1979.

R. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37:194–197, 1941.

E. Burke and J. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.

E. Burke, D. Elliman, and R. Weare. Specialised recombinative operators for timetabling problems. In *The Artificial Intelligence and Simulated Behaviour Workshop on Evolutionary Computing*, volume 993, pages 75–85. Springer, 1995.

E. Burke, D. Elliman, P. Ford, and R. Weare. Examination timetabling in British universities: A survey. In E Burke and P Ross, editors, *Practice and Theory of Automated Timetabling (PATAT) I*, volume 1153 of *LNCS*, pages 76–92. Springer, 1996.

H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos. Local search and constraint programming for the post enrolment-based course timetabling problem. In E. Burke and M. Gendreau, editors, *Practice and Theory of Automated Timetabling (PATAT) VII*, 2008.

H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos. Local search and constraint programming for the post enrolment-based timetabling problem. *Annals of Operational Research*, 194:111–135, 2012.

M. Caramia and P. Dell'Olmo. Solving the minimum weigted coloring problem. *Networks*, 38(2):88–101, 2001.

M. Carrasco and M. Pato. A multiobjective genetic algorithm for the class/teacher timetabling problem. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling (PATAT) III*, volume 2079 of *LNCS*, pages 3–17. Springer, 2001.

F. Carroll and R. Lewis. The "engaged" interaction: Important considerations for the HCI design and development of a web application for solving a complex combinatorial optimization problem. *World Journal of Computer Application and Technology*, 1(3):75–82, 2013.

M. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47:373–383, 1996.

S. Ceschia, L. Di Gaspero, and A. Schaerf. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers and Operational Research*, 39:1615–1624, 2012.

G. Chaitin. Register allocation and spilling via graph coloring. *SIGPLAN Not.*, 39 (4):66–74, 2004.

M. Chams, A. Hertz, and O. Dubuis. Some experiments with simulated annealing for coloring graphs. *European Journal of Operations Research*, 32:260–266, 1987.

P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-91*, pages 331–337, 1991.

M. Chiarandini and T. Stützle. An application of iterated local search to graph coloring. *Proceedings of the Computational Symposium on Graph Coloring and it's Generalizations*, pages 112–125, 2002.

M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432, 2006. ISSN 1094-6136. doi: http://dx.doi.org/10.1007/s10951-006-8495-8.

M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164(1):51–229, 2006.

E. Coffman, M. Garey, D. Johnson, and A. LaPaugh. Scheduling file transfers. *SIAM Journal of Computing*, 14(3):744–780, 1985.

C. Colbourn. The complexity of completing partial Latin squares. *Discrete Applied Mathematics*, 8(1):25–30, 1984.

E. Coll, G. Duran, and P. Moscato. A discussion on some design principles for efficient crossover operators for graph coloring problems. *Anais do XXVII Simposio Brasileiro de Pesquisa Operacional, Vitoria-Brazil*, 1995.

A. Colorni, M. Dorigo, and V Maniezzo. Metaheuristics for high-school timetabling. *Computational Optimization and Applications*, 9(3):277–298, 1997.

S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi: 10.1145/800157.805047.

T. Cooper and J. Kingston. The complexity of timetable construction problems. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling (PATAT) I*, volume 1153 of *LNCS*, pages 283–295. Springer, 1996.

T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

D. Corne, P. Ross, and H. Fang. Evolving timetables. In L. Chambers, editor, *The Practical Handbook of Genetic Algorithms*, volume 1, pages 219–276. CRC Press, 1995.

P. Cote, T. Wong, and R. Sabourin. Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In E. Burke and M. Trick, editors, *Practice and Theory of Automated Timetabling (PATAT) V*, volume 3616 of *LNCS*, pages 294–312. Springer, 2005.

D. Cranston and L. Rabern. Brooks' theorem and beyond. *Journal of Graph Theory*, 2014. doi: 10.1002/jgt.21847.

J. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. *American Mathematical Society: Cliques, Coloring, and Satisfiability – Second DIMACS Implementation Challenge*, 26:245–284, 1996.

D. de Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21:47–65, 1988.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2000.

F. della Croce and D. Oliveri. Scheduling the Italian football league: an ILP-based approach. *Computers and Operations Research*, 33(7):1963–1974, 2006.

F. della Croce, R. Tadei, and P. Asioli. Scheduling a round-robin tennis tournament under courts and players unavailability constraints. *Annals of Operational Research*, 92:349–361, 1999.

M. Demange, D. de Werra, J. Monnot, and V. Paschos. Time slot scheduling of compatible jobs. *Journal of Scheduling*, 10:111–127, 2007.

L. Di Gaspero and A. Schaerf. Multi-neighbourhood local search with application to course timetabling. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling (PATAT) IV*, volume 2740 of *LNCS*, pages 263–287. Springer, 2002.

L. Di Gaspero and A. Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms*, 5(1):65–89, 2006.

L Di Gaspero and A. Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13(2):189–207, 2007.

J. Dinitz, D. Garnick, and B. McKay. There are 526,915,620 nonisomorphic one-factorizations of K_{12}. *Journal of Combinatorial Designs 2*, 2:273–285, 1994.

M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimisation by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern*, 26(1):29–41, 1996.

R. Dorne and J.-K. Hao. A new genetic local search algorithm for graph coloring. In A. Eiben, T. Back, M. Schoenauer, and H. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN) V*, volume 1498 of *LNCS*, pages 745–754. Springer, 1998.

A. Dupont, A. Linhares, C. Artigues, D. Feillet, P. Michelon, and M. Vasquez. The dynamic frequency assignment problem. *European Journal of Operational Research*, 195:75–88, 2009.

K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem: description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming*, volume 2239 of *LNCS*, pages 580–585. Springer, 2001.

K. Easton, G. Nemhauser, and M. Trick. Solving the traveling tournament problem: A combined integer programming and constraint programming approach. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling (PATAT) IV*, volume 2740 of *LNCS*, pages 100–109. Springer, 2003.

J. Egeblad and D. Pisinger. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers and Operational Research*, 36(4):1026–1049, 2009.

A. Eiben, J. van der Hauw, and J. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

M. Elf, M. Junger, and G. Rinaldi. Minizing breaks by maximizing cuts. *Operations Research Letters*, 31(5):343–349, 2003.

E. Erben. A grouping genetic algorithm for graph colouring and exam timetabling. *Practice and Theory of Automated Timetabling (PATAT) III*, 2079:132–158, 2001.

P. Erdős. *Theory of Graphs and its Applications*, chapter Problem 9, page 159. Czech Acad. Sci. Publ., 1964.

B. Escoffier, J. Monnot, and V. Paschos. Weighted coloring: further complexity and approximability results. *Information Processing Letters*, 97(3):98–103, 2006.

E. Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley and Sons, 1998.

I. Finocchi, A. Panconesi, and R. Silvestri. An experimental analysis of simple, distributed vertex colouring algorithms. *Algorithmica*, 41:1–23, 2005.

C. Fleurent and J. Ferland. Allocating games for the NHL using integer programming. *Operations Research*, 41(4):649–654, 1993.

C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph colouring. *Annals of Operational Research*, 63:437–461, 1996.

H. Furmańczyk. *Graph Colorings*, chapter Equitable Coloring of Graphs, pages 35–54. American Mathematical Society, 2004.

P. Galinier and J.-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

P. Galinier and A. Hertz. A survey of local search algorithms for graph coloring. *Computers and Operations Research*, 33:2547–2562, 2006.

M. Garey and D. Johnson. The complexity of near-optimal coloring. *Journal of the Association for Computing Machinery*, 23(1):43–49, 1976.

M. Garey and D. Johnson. *Computers and Intractability - A guide to NP-completeness*. W. H. Freeman and Company, San Francisco, first edition, 1979.

M. Garey, D. Johnson, and H. So. An application of graph coloring to printed circuit testing. *IEEE Transactions on Circuits and Systems*, CAS-23:591–599, 1976.

B. Gendron, A. Hertz, and P. St-Louis. On edge orienting methods for graph coloring. *Journal of Combinatorial Optimization*, 13(2):163–178, 2007.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

A. Gyárfás and J. Lehel. On-line and first fit colourings of graphs. *Journal of Graph Theory*, 12:217–227, 1988.

A. Hajnal and E. Szemerédi. *Combinatorial Theory and it's Application*, chapter Proof of a Conjecture by P. Erdős, pages 601–623. North-Holland, 1970.

P. Hansen, M. Labbé, and D. Schindl. Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. *Discrete Optimization*, 6 (2):135 – 147, 2009.

R. Hassin and J. Monnot. The maximum saving partition problem. *Operations Research Letters*, 33:242–248, 2005.

P. Heawood. Map-colour theorems. *Quarterly Journal of Mathematics*, 24:332–338, 1890.

M. Henz, T. Muller, and S. Theil. Global constraints for round robin tournament scheduling. *European Journal of Operational Research*, 153:92–101, 2004.

H. Hernández and C. Blum. FrogSim: Distributed graph coloring in wireless ad hoc networks. *Telecommunication Systems*, 55:211–223, 2014.

A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

A. Herzberg and M. Murty. Sudoku squares and chromatic polynomials. *Notices of the AMS*, 54(6):708–717, 2007.

A. Hoffman. On eigenvalues and colorings of graphs. In *Graph Theory and Its Applications*, Proc. Adv. Sem., Math., page 7991, Research Center, Univ. of Wisconsin, Madison, WI, 1969, Academic Press, New York, 1970.

I. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10:718–720, 1981.

J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the Association for Computing Machinery*, 21:549–568, 1974.

R. Janczewski, M. Kubale, K. Manuszewski, and K. Piwakowski. The smallest hard-to-color graph for algorithm DSatur. *Discrete Mathematics*, 236:151–165, 2001.

S. Jat and S. Yang. A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. *Journal of Scheduling*, 14:617–637, 2011.

T. Jensen and B. Toft. *Graph Coloring Problems*. Wiley-Interscience, first edition, 1994.

A. Kanevsky. Finding all minimum-size separating vertex sets in a graph. *Networks*, 23:533–541, 1993.

M. Karp. *Complexity of Computer Computations*, chapter Reducibility Among Combinatorial Problems, pages 85–103. Plenum, New York, 1972.

M. Kearns, S. Suri, and N. Montfort. An experimental study of the coloring problem on human subject networks. *Science*, 313:824–827, 2006.

A. Kempe. On the geographical problem of the four colours. *American Journal of Mathematics*, 2:193–200, 1879.

G. Kendall, S. Knust, C. Ribeiro, and S. Urrutia. Scheduling in sports, an annotated bibliography. *Computers and Operations Research*, 37(1):1–19, 2010.

H. Kierstead and A. Kostochka. A short proof of the Hajnal-Szemerédi theorem on equitable coloring. *Combinatorics, Probability and Computing*, 17:265–270, 2008.

H. Kierstead and W. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

H. Kierstead, A. Kostochka, M. Mydlarz, and E. Szemerédi. A fast algorithm for equitable graph coloring. *Combinatorica*, 30:217–224, 2010.

T. Kirkman. On a problem in combinations. *Cambridge Dublin Math Journal*, 2: 191–204, 1847.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

V. Kolmogorov. Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

D. König. Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére. *Matematikai és Természettudományi Értestö*, 34:104–119, 1916.

S. Korman. *Combinatorial Optimization.*, chapter The Graph-Coloring Problem, pages 211–235. Wiley, New York, 1979.

P. Kostuch. The university course timetabling problem with a 3-phase approach. In E. Burke and M. Trick, editors, *Practice and Theory of Automated Timetabling (PATAT) V*, volume 3616 of *LNCS*, pages 109–125. Springer, 2005.

M. Kubale and B. Jackowski. A generalized implicit enumeration algorithm for graph coloring. *Commun. ACM*, 28(28):412–418, 1985.

K. Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

M. Laguna and R. Marti. A grasp for coloring sparse graphs. *Computational Optimization and Applications*, 19:165–78, 2001.

F. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.

R. Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers and Operations Research*, 36(7):2295–2310, 2009.

R. Lewis. A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. *Annals of Operational Research*, 194(1):273–289, 2012.

R. Lewis. *Springer Handbook of Computational Intelligence*, chapter Graph Coloring and Recombination, pages 1239–1254. Studies in Computational Intelligence. Springer, 2015.

R. Lewis and B. Paechter. Finding feasible timetables using group based operators. *IEEE Transactions on Evolutionary Computation*, 11(3):397–413, 2007.

R. Lewis and J. Thompson. On the application of graph colouring techniques in round-robin sports scheduling. *Computers and Operations Research*, 38(1):190–204, 2010.

R. Lewis and J. Thompson. Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, 240:637–648, 2015.

R. Lewis, J. Thompson, C. Mumford, and J. Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers and Operations Research*, 39(9):1933–1950, 2012.

A. Lim, B. Rodrigues, and X. Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research*, 174(3):1459–1478, 2006.

L. Lovász, M. Saks, and W. Trotter. An on-line graph colouring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75:319–325, 1989.

Z. Lü and J.-K Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241 – 250, 2010a.

Z. Lü and J-K. Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010b.

D. MacKenzie. Graph theory uncovers the roots of perfection. *Science*, 38:297, 2002.

E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008.

E. Malaguti, M. Monaci, and P. Toth. Models and heuristic algorithms for a weighted vertex coloring problem. *Journal of Heuristics*, 15:503–526, 2009.

E. Malaguti, M. Monaci, and P. Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

B. McCollum, A. Schaerf, B.. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Di Gaspero, R. Qu, and E. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.

C. McDiarmid and B. Reed. Channel assignment and weighted coloring. *Networks*, 36(2):114–117, 2000.

G. McGuire, B. Tugemann, and G. Civario. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem. *Computing Research Repository*, abs/1201.0749, 2012.

A. Mehrotra and M. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

A. Mehrotra and M. Trick. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies. Operations Research/Computer Science Interfaces Series Volume 37*, chapter A Branch-and-Price Approach for Graph Multi-coloring, pages 15–29. Springer, 2007.

I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156:159–179, 2008.

W. Meyer. Equitable coloring. *American Mathematical Monthly*, 80:920–922, 1973.

J. Misra and D. Gries. A constructive proof of Vizing's theorem. *Information Processing Letters*, 41:131–133, 1992.

R. Miyashiro and T. Matsui. A polynomial-time algorithm to find an equitable home-away assignment. *Operations Research Letters*, 33:235–241, 2005.

R. Miyashiro and T. Matsui. Minimizing the carry-over effects value in a round-robin tournament. In E. Burke and H. Rudova, editors, *PATAT '06 Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, pages 460–464, 2006a.

R. Miyashiro and T. Matsui. Semidefinite programming based approaches to the break minimization problem. *Computers and Operations Research*, 33(7):1975–1992, 2006b.

J. Moody and D. White. Structural cohesion and embeddedness: A hierarchical concept of social groups. *American Sociological Review*, 68(1):103–127, 2003.

C. Morgenstern and H. Shapiro. Coloration neighborhood structures for general graph coloring. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 226–235, San Francisco, California, USA, Society for Industrial and Applied Mathematics, 1990.

T. Müller and H. Rudova. Real life curriculum-based timetabling. In D Kjenstad, A. Riise, T. Nordlander, B. McCollum, and E. Burke, editors, *Practice and Theory of Automated Timetabling (PATAT 2012)*, pages 57–72, 2012.

C. Mumford. New order-based crossovers for the graph coloring problem. In *Parallel Problem Solving from Nature (PPSN) IX*, volume 4193 of *LNCS*, pages 880–889. Springer, 2006.

J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

J. Mycielski. Sur le coloriage des graphes. *Colloquium Mathematicum*, 3:161–162, 1955.

G. Nemhauser and M. Trick. Scheduling a major college basketball conference. *Operations Research*, 46:1–8, 1998.

J. Nielsen. The need for web design standards. Online Article: www.nngroup.com/-articles/the-need-for-web-design-standards, September 2004.

C. Nothegger, A. Mayer, A. Chwatal, and G. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operational Research*, 194:325–339, 2012.

L. Ouerfelli and H. Bouziri. Greedy algorithms for dynamic graph coloring. In *Proceedings of the International Conference on Communications, Computing and Control Applications (CCCA)*, pages 1–5, 2011. doi: 10.1109/CCCA.2011.6031437.

B. Paechter, R. Rankin, A. Cumming, and T. Fogarty. Timetabling the classes of an entire university with an evolutionary algorithm. In T. Baeck, A. Eiben, M. Schoenauer, and H. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN) V*, volume 1498 of *LNCS*, pages 865–874. Springer, 1998.

L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, volume 2279 of *LNCS*, pages 121–130. Springer, 2002.

M. Pelillo. *Encyclopedia of Optimization*, chapter Heuristics for Maximum Clique and Independent Set, pages 1508–1520. Springer, 2nd edition, 2009.

C. Perea, J. Alcaca, V. Yepes, F. Gonzalez-Vidosa, and A. Hospitaler. Design of reinforced concrete bridge frames by heuristic optimization. *Advances in Engineering Software*, 39(8):676–688, 2008.

S. Petrovic and Y. Bykov. A multiobjective optimisation approach for exam timetabling based on trajectories. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling (PATAT) IV*, volume 2740 of *LNCS*, pages 181–194. Springer, 2003.

D. Porumbel, J.-K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37:1822–1832, 2010.

B. Reed. A strengthening of Brooks' theorem. *Journal of Combinatorial Theory, Series B*, 76(2):136–149, 1999.

C. Ribeiro and S. Urrutia. Heuristics for the mirrored travelling tournament problem. *European Journal of Operational Research*, 179(3):775–787, 2007.

A. Rizzoli, R. Montemanni, E. Lucibello, and L. Gambardella. Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1(2):135–151, 2007.

N. Robertson, D. Sanders, P. Seymour, and R. Thomas. The four color theorem. *Journal of Combinatorial Theory, Series B*, 70:2–44, 1997.

D. Rose, G. Lueker, and R. Tarjan. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of Computing*, 5(2):266–283, 1976.

P. Ross, E. Hart, and D. Corne. Genetic algorithms and timetabling. In A. Ghosh and K. Tsutsui, editors, *Advances in Evolutionary Computing: Theory and Applications*, Natural Computing, pages 755–771. Springer, 2003.

O. Rossi-Doria, M. Samples, M. Birattari, M. Chiarandini, J. Knowles, M. Manfrin, M. Mastrolilli, L. Paquete, B. Paechter, and T. Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling (PATAT) IV*, volume 2740 of *LNCS*, pages 329–351. Springer, 2002.

E. Russell and F. Jarvis. There are 5,472,730,538 essentially different Sudoku grids. Online Article: www.afjarvis.staff.shef.ac.uk/sudoku/sudgroup.html, September 2005.

K. Russell. Balancing carry-over effects in round-robin tournaments. *Biometrika*, 67(1):127–131, 1980.

R. Russell and J. Leung. Devising a cost effective schedule for a baseball league. *Operations Research*, 42(4):614–625, 1994.

A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13 (2):87–127, 1999.

S. Sekiner and M. Kurt. A simulated annealing approach to the solution of job rotation scheduling problems. *Applied Mathematics and Computation*, 188(1): 31–45, 2007.

K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers and Operations Research*, 45:12–24, 2014.

J. Spinrad and G. Vijayan. Worst case analysis of a graph coloring algorithm. *Discrete Applied Mathematics*, 12:89–92, 1984.

J. Thompson and K. Dowsland. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156:313–324, 2008.

M. Trick. A schedule-then-break approach to sports timetables. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling (PATAT) III*, volume 2079 of *LNCS*, pages 242–253. Springer, 2001.

J. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9: 63–82, 1988.

C. Valenzuela. A study of permutation operators for minimum span frequency assignment using an order based representation. *Journal of Heuristics*, 7:5–21, 2001.

J. van den Broek and C. Hurkens. An IP-based heuristic for the post enrolment course timetabling problrm of the ITC2007. *Annals of Operational Research*, 194:439–454, 2012.

P. van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, 1987.

V. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz.*, 3: 25–30, 1964.

R. Wilson. *Four Colors Suffice: How the Map Problem Was Solved*. Penguin Books, 2003.

L. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.

M. Wright. Timetabling county cricket fixtures using a form of tabu search. *Journal of the Operational Research Society*, 47(7):758–770, 1994.

M. Wright. Scheduling fixtures for Basketball New Zealand. *Computers and Operations Research*, 33(7):1875–1893, 2006.

Q. Wu and J-K. Hao. Coloring large graphs based on independent set extraction. *Computers and Operational Research*, 39:283–290, 2012.

T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A:1052–1060, 2003.

Index

© Springer International Publishing Switzerland 2016

R.M.R. Lewis, *A Guide to Graph Colouring*,
DOI 10.1007/978-3-319-25730-3