

AIT608 Machine Learning - Assignment

Deadline 15/4/2010

Dr Dimitris C. Dracopoulos
Email: d.dracopoulos@westminster.ac.uk

Description

Design and implement a genetic algorithm approach in Matlab (or Java) for the graph colouring problem.

A graph is a set of points called *nodes* with connections between some pairs of nodes. A graph does not have to be fully connected, i.e. there are pairs of nodes which are not connected to each other. Associated with each node there is a value called *weight*. An example of a graph is illustrated in Figure 1.

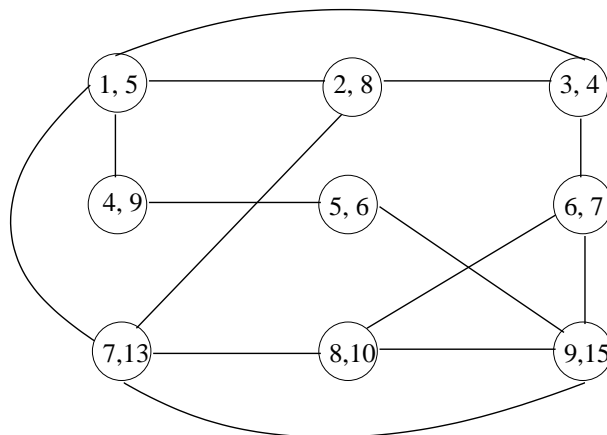


Figure 1: *Graph1*. An example of a graph. Circles indicate nodes and lines indicate links. The first number in each node is its index and the second its weight.

The graph colouring problem is defined as follows: given a graph with a weight on each node, and given n colours, the problem is to achieve the highest score by:

1. Assign any of the nodes on the graph a colour from the set of n colours if you like, but
2. No pair of nodes connected by a link can have the same colour.
3. Your score is the total weight of the coloured nodes.

The purpose of the Genetic Algorithm implementation is to find a solution to the problem, by choosing a colouring scheme which maximises the sum of the weights of the coloured nodes (uncoloured nodes are not included in the sum).

When one colour is used for the graph colouring problem of Figure 1 (i.e. $n = 1$), the optimum solution would be to colour nodes 9, 4, 2 and leave the rest of the nodes uncoloured resulting in a score of $15 + 9 + 8 = 32$. With two colours, the best solution would be to assign the first colour to nodes 9, 4 and 2 and the second colour to nodes 7, 6 and 5.

When one colour is used for the graph colouring problem of Figure 2, the optimum solution is to colour nodes 1, 5 and 3 to obtain a score of $12 + 9 + 14 = 35$. With two colours, the best approach would be to assign the first colour to nodes 1, 5 and 3, and the second colour to nodes 2, 4 and 6.

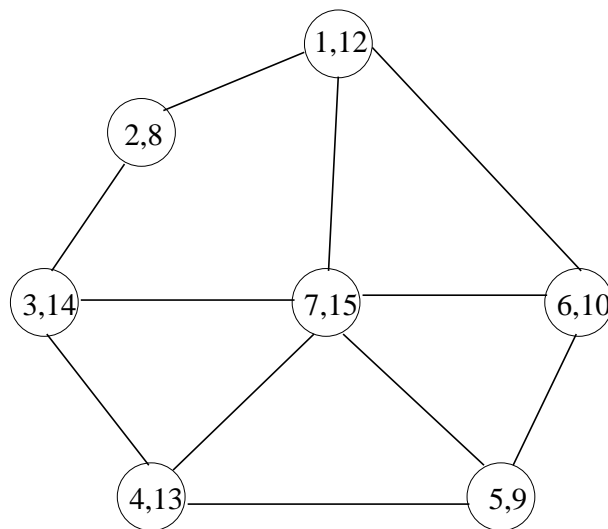


Figure 2: *Graph2*. Another example of a graph.

Design a Genetic Algorithm approach by defining all the appropriate genetic operations (e.g. crossover) and parameters used in a genetic algorithm (e.g. population size, probability of operations, etc.). The genetic algorithm should consider and produce valid colouring schemes (i.e. no solution should be considered where two coloured nodes are linked together and have the same colour. Two nodes linked to each other can be left without any colour though.)

Your implementation should be generic enough so as to accept as input an arbitrary graph and an arbitrary number of colours n . For this reason your implementation must be able to read any graph from a file following the format which is described in the test cases `graph1.dat`, `graph2.dat`, `graph3.dat` below.

You can make your own assumptions for any details not found in the specification of the problem.

You cannot use an existing implementation of genetic algorithms, but you must create your own implementation in either Matlab or Java and solve the above problem.

You should test your approach on the two graphs illustrated in Figures 1 and 2 and a larger graph consisting of 100 nodes which are defined by the following input files respectively:

- <http://users.wmin.ac.uk/~dracopd/DOCUM/courses/2ait608/coursework/graph1.dat>
- <http://users.wmin.ac.uk/~dracopd/DOCUM/courses/2ait608/coursework/graph2.dat>
- <http://users.wmin.ac.uk/~dracopd/DOCUM/courses/2ait608/coursework/graph3.dat>

The first line is of the format:

`p NODES EDGES`

`p` indicates this is the problem line, and the two numbers which follow define the number of nodes and edges in the graph. For example, in `graph1.dat` there are 9 nodes and 14 edges.

Following the first line, there is a number of lines starting with an 'e' each one followed by a pair of numbers. Such a line, describes an edge (u, v) which links nodes u and v . Note that each edge (u, v) appears exactly once in a file and it is not repeated as (v, u) . In the first graph `graph1.dat` (corresponding to the data of Figure 1), the first edge is the link between nodes 1 and 2.

Following the series of lines describing edges (lines which start with an 'e'), there is a series of lines starting with the character 'n', followed by two numbers. These define nodes and their weights. The first number indicated a node number and the second its associated weight. For example, in `graph1.dat` the first line starting with an 'n' is `n 1 5` which indicates that node 1 has a weight with the value of 5.

Your implementation should be tested with all three graphs above and by using both one and two colours ($n = 1$, $n = 2$) for each data file, i.e. in total you have 6 test cases. For the graphs of Figures 1, 2 you can compare the solution of your genetic algorithm with the optimum which was given above.

For the larger problem, you can compare the solution found by your genetic algorithm with the solution produced by the greedy algorithm. To find the solution produced by the greedy algorithm, order the nodes of the graph according to the descending order of their weights. Then starting with the node having the highest weight, colour it, and colour each subsequent node in the sorted node list assuming this is valid (the current node i under consideration should be coloured only if there is no node j previously visited with the same colour and there is a link between i and j).

A viva for each student based on his/her submission will take place on 16/4/2010, during the tutorial session. Failure to turn up in the viva will result in a zero (0) mark for the whole assignment.

Files to submit

You should submit all the files with source code and a technical report (in PDF format) describing your solution. The maximum size of the technical report is five pages (including diagrams).

Deadline: April 15th 2010.

Submission Instructions

You should submit via BlackBoard's Assignment functionality (do NOT use email, as email submissions will be ignored.), the zip file containing all the files described above in the "Files to submit" section. Source code should be in plain ASCII format.

Note that Blackboard will allow to make a submission **ONLY ONCE**. So make sure before submitting, that all the files you want to submit are contained there (or in the zip file you submit). Submissions with missed files cannot be resubmitted, so do **NOT** email me with such requests.

The following describes how to submit your work via BlackBoard:

1. Access <http://learning.wmin.ac.uk> and login using your username and password (if either of those is not known to you, ask the HelpDesk at the Library.).
2. Click on the module's name, **MODULE: 2AIT608.2010.2 MACHINE LEARNING AND DATA MINING** found under **My Modules & Courses**.
3. Click on the **Assignments** button found on the left hand side menu.
4. Click on **View/Complete Assignment**.
5. Attach your files containing in your submission, by using the **Browse** button repeatedly. Note that all your files must have an extension, otherwise you will not be able to upload them.
6. Fill in the requested information:
 - *Comments*: Type your full name and your registration number, followed by:
"I confirm that I understand what plagiarism is and have read and understood the section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely my own. Any work from other authors is duly referenced and acknowledged."
7. Click the **Submit** button.

If Blackboard is unavailable before the deadline you must email me before the deadline with a copy of the assignment, following the naming, title and comments conventions as given above and stating the time that you tried to access Blackboard. You are still expected to submit your assignment via Blackboard. Please keep checking Blackboard's availability at regular intervals up to and after the deadline for submission. You must submit your coursework through Blackboard as soon as you can after Blackboard becomes available again even if you have also emailed the coursework to me.