Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code: CM3109
Module Title: Combinatorial Optimisation
Lecturer: Richard Booth
Assessment Title: Coursework
Assessment Number: 1
Date Set: 12/11/2021
Submission Date and Time: 10/12/21 at 9:30am
Return Date: 19/1/2022

This assignment is worth 30% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1. If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;
2. If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Your submission must include the official Coursework Submission Cover sheet, which can be found here:

https://docs.cs.cf.ac.uk/downloads/coursework/Coversheet.pdf

Submission Instructions

You are required to complete 3 tasks about implementing a simulated annealing algorithm, as described in detail in the attachment. The answers to tasks 1 and 3 should be submitted as a single pdf or Word file. The answer to task 2 should be submitted as source code in the programming language you choose to use.

Description		Type	Name
Cover sheet	**Compulsory**	One PDF (.pdf) file	[student number].pdf
Task 1+3	**Compulsory**	One PDF (.pdf) or Word file (.doc or .docx)	Text_[student number].pdf/doc/docx
Task 2	**Compulsory**	One or more source code files in programming language of choice	No restriction

Any code submitted will be run on a system equivalent to those available in the Linux laboratory and must be submitted as stipulated in the instructions above.

Any deviation from the submission instructions above (including the number and types of files submitted) will result in a mark of zero for the assessment or question part.

Staff reserve the right to invite students to a meeting to discuss coursework submissions

---

Assignment

The assignment is described in detail in the attachment.

Learning Outcomes Assessed

Explain the concepts of problem state spaces and search

Criteria for assessment

Credit will be awarded against the following criteria.

- [Correctness] Do the answers correctly address the requirements of each task? Does the provided code run correctly in Task 2?
- [Clarity] Are explanations and summaries easily understandable? Is documentation clear? Are comments provided in the code to clarify non-standard code fragments?
- [Understanding of concepts] Do the answers show an understanding of basic optimisation concepts such as neighbourhood?

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned 19/1/2022 via Learning Central. This will be supplemented with oral feedback via individual meetings.

CM3109 Coursework
Autumn semester 2021

Notes before starting

- The following programming exercises may be done using a programming language of your own choice, though all parts should be done using the same language. If you wish to use anything other than Java or Python then please let me know well in advance so I can make sure I have everything necessary to run your program installed on my computer.
- If you use any external sources for solving your coursework (e.g. pseudocode from a website), you should add a comment with a reference to these sources and a brief explanation of how they have been used.

The aim of this coursework is to use simulated annealing to solve an optimisation problem coming from the field of *computational social choice* - a hot interdisciplinary topic currently receiving a lot of attention from researchers in several different fields, including AI. Section 1 provides necessary background to the problem, with the specific coursework tasks coming in Section 2.

1 Background

We start by introducing the problem we're looking at, then we discuss how to represent the input and outputs to this problem, and then we introduce a particular proposal for approaching this problem which we will aim to implement via simulated annealing.

The problem

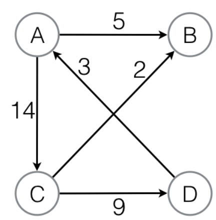
Suppose we have a competition with n participants, which is decided by playing a *tournament* in which each participant plays one match against each of the others. (These participants could be football teams, chess players, or anything). After the tournament is finished we know both *(i)* the winner of each match (or if it was a draw), and *(ii)* the margin of victory (represented by an integer greater than 0 if it wasn't a draw). The question is, based on this information, which participant should be judged to be the winner of the tournament, who is the runner-up, who places third, and so on up to the worst? In other words, which *ranking* of the participants (from best to worst) appropriately reflects the relative quality of the participants? (We do not allow ties in the ranking).

Weighted tournament graphs and rankings

We can picture such a tournament T as a directed graph whose nodes are the participants, and such that there exists a directed edge *from* i *to* j if i is the winner of the match "i versus j". Each such edge is labelled with a *weight* $w(i, j)$, which is the number representing the margin of victory. One example of such a weighted tournament graph is on the left of Fig. 1. (Note the lack of edge between B and D, indicating their match was drawn.) There is an edge from A to C labelled with 14, indicating that A was the very clear winner (by margin 14) of the match "A versus C". A also defeated B by a margin of 5, but was defeated by D by a margin of 3. These three results would suggest that D has claims to being declared the best, but D was itself defeated quite convincingly by C (by margin of 9), so it's not clear-cut.

Another way to represent a weighted tournament graph (more amenable for representation in a computer) is as an $n \times n$ matrix $[a_{ij}]$ whose rows and columns represent the participants $\{i \mid i = 1, \ldots, n\}$ and whose (i, j) entry a_{ij} equals 0 if i did not defeat j, and gives the weight $w(i, j)$ if i defeated j. For example, the matrix on the right of Fig. 1 is the matrix representation of the tournament graph to its left.

Concerning the output of our problem, i.e., rankings, the easiest way to represent a ranking is as a list $[i_1, i_2, \ldots, i_n]$ with the first entry i_1 representing the best participant, the second i_2 representing the second best, and so on. For instance, continuing the above example, the ranking which places D first followed by A, B and C, in that order, would be represented as $[D, A, B, C]$.



	A	B	C	D
A	0	5	14	0
B	0	0	0	0
C	0	2	0	9
D	3	0	0	0

Figure 1: An example tournament graph with $n = 4$ participants A, B, C, D on the left, with its matrix representation on the right

Kemeny rankings

In the above problem, we have to choose the ranking that "best fits" the results from a given weighted tournament T. Several different proposals have been made for how to precisely formalise what "best fit" means. We are going to focus on just one of them, which is based on the intuition of minimising the *disagreements* with T. Let's say a particular ranking R *disagrees with* T *on an edge* (x, y) if x defeats y in T but y is ranked above x in R. We can then measure how well R fits T by adding up all the weights of all the edges for which R disagrees with T. This results in the *Kemeny score*[1] of R (with respect to T), denoted by $c(R, T)$ and given by the following formula:

$$c(R, T) = \sum \{w(x, y) \mid R \text{ disagrees with } T \text{ on } (x, y)\}.$$

For example if T is the weighted tournament in Fig. 1 and $R = [D, A, B, C]$ then R disagrees with T on (C, D) and (C, B), which have weights 9 and 2 respectively. Hence $c(R, T) = 9 + 2 = 11$. The ranking that best fits T is then taken to be any ranking R that has the lowest possible Kemeny score. Thus we arrive at the following optimisation problem:

Optimisation problem. Given a weighted tournament T, find a ranking R that minimises the Kemeny score $c(R, T)$.

A ranking that minimises the Kemeny score (with respect to a given tournament T) is called a *Kemeny ranking* (with respect to T). Note that

[1] Named after John Kemeny.

some tournaments can have more than one Kemeny ranking. The tournament in Fig. 1 has two Kemeny rankings, namely $[A, C, B, D]$ and $[A, C, D, B]$.

2 Coursework questions

The tasks below require you to write a program (in a programming language of your choice) that is able to read a specific weighted tournament from a separate file (to be described below), and return a Kemeny ranking for this tournament - or at least an approximation of one - using simulated annealing (see Week 5 Presentation Slides on Learning Central for the pseudocode of the simulated annealing algorithm).

This coursework will use a particular weighted tournament containing real data (`Formula_One_1984.wmg`) from the 1984 World Formula One Motor Racing Championship, for which there were a total of 35 participants. This file, along with an explanation of its data format (see `How_to_read_tournament_data.pdf`) can be downloaded from Learning Central.

First part: Setting SA parameters

As discussed during lecture, simulated annealing has a number of parameters that need to be fixed by the programmer in advance (see Week 5 Learning Materials). Some will be fixed below, but for others you will be free to experiment.

- ***initial solution.*** For the initial solution you must always use the ranking $[1, 2, 3, 4, \ldots, 34, 35]$, using the numbering of candidates specified in `Formula_One_1984.wmg`.

- ***cooling schedule.*** For initial temperature TI and temperature length TL you are free to experiment with different values. Your only restriction regarding cooling ratio f is that $f(T)$ should be of the form $f(T) = a \times T$, where a is some fixed parameter between 0 and 1 (which you are free to choose).

- ***cost function.*** The cost of a solution (i.e., possible ranking) R is the Kemeny score $c(R, T)$, defined in Section 1 above, where T is the tournament loaded from `Formula_One_1984.wmg`.

- ***stopping criterion.*** The algorithm should STOP if a pre-specified number `num_non_improve` of solutions have been looked at without improving on the current best solution. You are free to experiment with different values of `num_non_improve`.

The only parameter not yet specified is the neighbourhood function, which brings us to...

TASK 1. Write down a definition of a *neighbourhood* N that is specific to this problem, i.e., specify the conditions for when any given ranking R_2 is a neighbour of another R_1, and illustrate your definition with an example. Justify your choice of definition by showing how the cost of a neighbouring solution R_2 can easily be computed from the cost of current solution R_1.

(Total 5 marks)

Indicative mark breakdown is as follows:

1st	≥ 3.5	Clear, unambiguous definition and example, with clear justification
2.1	≥ 3	Clear, unambiguous definition given with example
2.2	≥ 2.5	Understandable definition given with example
3rd	≥ 2	Definition given that is understandable
Fail	< 2	Minimal requirements for 3rd not met

Second part: Implementation

After having set up the parameters, you must now implement the algorithm.

TASK 2. Write a program implementing the simulated annealing algorithm to find a ranking that minimises the Kemeny score with respect to the given weighted tournament T specified in `Formula_One_1984.wmg`. You must use the parameters in accordance with the first part above, and use the neighbourhood you defined in Task 1. Your program should have the following behaviour.

1. It should be executable via the command line, taking the file `Formula_One_1984.wmg` as an argument.

2. After running, your program should print to the screen the following information:

   - The best ranking it found (positions 1 to 35), in the form of a table, using the real names of the candidates, which are included in `Formula_One_1984.wmg`.
   - The Kemeny score of this best ranking.
   - The algorithm's runtime in milliseconds.

Your code will be judged on whether the SA algorithm is correctly implemented and, given it is correctly implemented, whether the solution returned is optimal (or at least close to optimal) and is returned within reasonable runtime. *You must highlight in your code (e.g., with comments) the part that deals with the actual steps of the SA algorithm. Failure to do so may lead to loss of marks.*

(Total 10 marks)

Indicative mark breakdown is as follows:

1st	≥ 7	Program implements SA correctly and returns optimal solution almost instantaneously
2.1	≥ 6	Program implements SA correctly and returns optimal solution eventually, or returns a near optimal solution almost instantaneously
2.2	≥ 5	Program produces output and implements SA correctly. SA steps in the code are adequately highlighted via code comments
3rd	≥ 4	Program produces required output and implements SA correctly, but makes minor errors or lacks code comments
Fail	< 4	Program doesn't print to the screen the required outputs, or prints only hard-coded outputs, or contains major errors

Third part: Selecting best parameters and discussion

As stated in the first part, you are free to try out different values for the parameters TI, TL and the "a" in $f(T) = a \times T$, as well as the number `num_non_improve` in the stopping criterion. The final task is about experimentally tuning to the best choice of parameters and reflecting on your results.

TASK 3. Give the values of the parameters TL, TI and a (in $f(T) = a \times T$), as well as `num_non_improve`, which seem to give the best solutions for your program. For this "best" choice of parameters provide screenshots of the output of your program. Write a short summary (max 300 words) of your results, indicating the range of different values you tried for the parameters, which parameters' variation had the biggest effect on the output solution, etc. Extra marks available for deeper analysis and presentation/visualisation of results, e.g., via use of graphs showing results of varying the different SA parameters Also offer some speculation on the presence of local optima in this problem. Are there many? **(Total 10 marks)**

Indicative mark breakdown is as follows:

1st	≥ 7	Varied visualisations provided with clear and justified conclusions plus deep understanding evidenced by, e.g., clear discussion on the presence of local optima.
2.1	≥ 6	Varied visualisations provided with clear conclusions offered on the information contained within.
2.2	≥ 5	Some minimal presentation/visualisation of results provided
3rd	≥ 4	Screenshot showing best parameters provided. Brief discussion on the range of different values tried and on which parameters had the biggest effect on the output.
Fail	< 4	Minimal requirements for 3rd not met