C1424548

CM3109: Combinatorial Optimisation

Professor David Walker

Combinatorial Optimisation Coursework

December 14, 2017

Contents

1 Neighbourhood Definition

In my implementation, a ranking R_1 is a neighbour of a second ranking R_2 if and only if R_1 can be transformed into R_2 by removing a single participant from and reinserting it at some position.

The initial and new positions could be the same because rankings are self-neighbouring, but in my implementation, I exclude these cases during random neighbour generation.

Any other ranking can be reached via a series of neighbourhood moves. To reach any given ranking from any other, simply shift each participant, one at a time, into their ranks in the target ranking. With this scheme, any ranking with n participants will have $(n-1)^2$ neighbours.

1.1 Example

The ranking $[D, C, B, A]$ has 9 neighbours:

$$[D, C, \underline{A}, \underline{B}]\ [D, \underline{A}, C, B]\ [C, B, A, \underline{D}]$$
$$[D, \underline{B}, \underline{C}, A]\ [\underline{B}, D, C, A]\ [D, B, A, \underline{C}]$$
$$[\underline{C}, \underline{D}, B, A]\ [\underline{A}, D, C, B]\ [C, B, \underline{D}, A]$$

On each neighbour I have underlined the participants that could have been shifted to get to that neighbour.

There are two ways to reach the neighbours that only require shifting one place to the left or right. I have compensated for this in my random neighbour generation by regenerating the insertion location 50% of the time in these cases.

1.2 Iterative Cost Calculation

With this neighbourhood scheme, the Kemeny score of a neighbour can be calculated without evaluating every edge in the graph. This can be done without computing the new ranking.

To calculate the Kemeny score of a neighbouring ranking where the participant at rank R_1 has been shifted to rank R_2:

1. *Start the new Kemeny score at the value of your current Kemeny score.*
2. *Using your current ranking, take graph edges between the participant at R_1 and the participants from R_1 to R_2 (excluding R_1, including R_2).*

   These are all the edges for which agreement/disagreement with the results will swap.
3. *For each edge:*
   - *If it disagrees with your ranking, subtract its weight from the new Kemeny score (it will no longer disagree in the neighbouring ranking).*
   - *If it agrees with your ranking, add its weight to the new Kemeny score (it will disagree in the neighbouring raking).*

2 Implementation

My implementation is written in Java across 5 source files:

2.1 Application.java

Contains the constants specifying my values for each parameter.
Coordinates the other classes.
Prints the output to the console.

2.2 Ranking.java

Represents a possible order to rank participants.
Contains the code to generate random neighbouring rankings.
Contains the code to evaluate Kemeny score from scratch.

2.3 TournamentResults.java

Represents the weighted tournament graph.
Allows looking up the weights of specific matchups.
Maps between numerical ids and participant names.

2.4 SimulatedAnnealingService.java

Contains the code to optimise a given initial solution using simulated annealing with specified parameters. Also keeps track of uphill moves made for use in the text output.

2.5 WMGParsingService.java

Contains a single static method for parsing the contents of the supplied *.wmg* file. The method returns a TournamentResults object.

3 Simulated Annealing Parameters

After some experimentation, I have found the following values to be optimal for my implementation:

$initial\_temperature = 1$
$temperature\_length = 10$
$cooling\_ratio = 0.95$
$num\_non\_improve = 8000$

3.1 Screenshots

The following are screenshots of the output of my program on 5 consecutive runs. The screenshots are taken of the console window in JetBrains IntelliJ IDEA.

```
KEMENY SCORE:  62
RUNTIME:       18ms
UPHILL MOVES:  1

POS  NAME
  1  Niki Lauda
  2  Alain Prost
  3  Rene Arnoux
  4  Elio de Angelis
  5  Corrado Fabi
  6  Michele Alboreto
  7  Derek Warwick
  8  Nelson Piquet
  9  Patrick Tambay
 10  Andrea de Cesaris
 11  Mauro Baldi
 12  Thierry Boutsen
 13  Teo Fabi
 14  Riccardo Patrese
 15  Gerhard Berger
 16  Jo Gartner
 17  Nigel Mansell
 18  Keke Rosberg
 19  Ayrton Senna
 20  Eddie Cheever
 21  Marc Surer
 22  Jonathan Palmer
 23  Martin Brundle
 24  Huub Rothengatter
 25  Jacques Laffite
 26  Stefan Bellof
 27  Francois Hesnault
 28  Stefan Johansson
 29  Piercarlo Ghinzani
 30  Manfred Winkelhock
 31  Johnny Cecotto
 32  Philippe Streiff
 33  Philippe Alliot
 34  Pierluigi Martini
 35  Mike Thackwell
```

```
KEMENY SCORE:  62
RUNTIME:       19ms
UPHILL MOVES:  1

POS  NAME
  1  Niki Lauda
  2  Alain Prost
  3  Elio de Angelis
  4  Rene Arnoux
  5  Corrado Fabi
  6  Derek Warwick
  7  Michele Alboreto
  8  Nelson Piquet
  9  Patrick Tambay
 10  Andrea de Cesaris
 11  Mauro Baldi
 12  Thierry Boutsen
 13  Teo Fabi
 14  Riccardo Patrese
 15  Gerhard Berger
 16  Jo Gartner
 17  Nigel Mansell
 18  Keke Rosberg
 19  Ayrton Senna
 20  Eddie Cheever
 21  Marc Surer
 22  Jonathan Palmer
 23  Martin Brundle
 24  Huub Rothengatter
 25  Jacques Laffite
 26  Stefan Bellof
 27  Francois Hesnault
 28  Stefan Johansson
 29  Piercarlo Ghinzani
 30  Manfred Winkelhock
 31  Philippe Streiff
 32  Johnny Cecotto
 33  Philippe Alliot
 34  Pierluigi Martini
 35  Mike Thackwell
```

```
KEMENY SCORE:  62
RUNTIME:       17ms
UPHILL MOVES:  0

POS  NAME
  1  Alain Prost
  2  Niki Lauda
  3  Elio de Angelis
  4  Rene Arnoux
  5  Corrado Fabi
  6  Derek Warwick
  7  Michele Alboreto
  8  Nelson Piquet
  9  Patrick Tambay
 10  Andrea de Cesaris
 11  Mauro Baldi
 12  Teo Fabi
 13  Thierry Boutsen
 14  Riccardo Patrese
 15  Gerhard Berger
 16  Jo Gartner
 17  Nigel Mansell
 18  Keke Rosberg
 19  Ayrton Senna
 20  Eddie Cheever
 21  Marc Surer
 22  Jonathan Palmer
 23  Martin Brundle
 24  Huub Rothengatter
 25  Jacques Laffite
 26  Stefan Bellof
 27  Francois Hesnault
 28  Stefan Johansson
 29  Piercarlo Ghinzani
 30  Manfred Winkelhock
 31  Johnny Cecotto
 32  Philippe Streiff
 33  Philippe Alliot
 34  Pierluigi Martini
 35  Mike Thackwell
```

```
KEMENY SCORE:  62
RUNTIME:       33ms
UPHILL MOVES:  1

POS  NAME
  1  Alain Prost
  2  Niki Lauda
  3  Elio de Angelis
  4  Rene Arnoux
  5  Corrado Fabi
  6  Derek Warwick
  7  Michele Alboreto
  8  Nelson Piquet
  9  Patrick Tambay
 10  Andrea de Cesaris
 11  Mauro Baldi
 12  Teo Fabi
 13  Thierry Boutsen
 14  Riccardo Patrese
 15  Gerhard Berger
 16  Jo Gartner
 17  Nigel Mansell
 18  Keke Rosberg
 19  Ayrton Senna
 20  Eddie Cheever
 21  Marc Surer
 22  Jonathan Palmer
 23  Martin Brundle
 24  Huub Rothengatter
 25  Jacques Laffite
 26  Stefan Bellof
 27  Francois Hesnault
 28  Stefan Johansson
 29  Piercarlo Ghinzani
 30  Manfred Winkelhock
 31  Johnny Cecotto
 32  Philippe Streiff
 33  Philippe Alliot
 34  Pierluigi Martini
 35  Mike Thackwell
```

```
KEMENY SCORE:  63
RUNTIME:       18ms
UPHILL MOVES:  4

POS  NAME
  1  Alain Prost
  2  Niki Lauda
  3  Rene Arnoux
  4  Elio de Angelis
  5  Corrado Fabi
  6  Michele Alboreto
  7  Derek Warwick
  8  Nelson Piquet
  9  Patrick Tambay
 10  Andrea de Cesaris
 11  Mauro Baldi
 12  Thierry Boutsen
 13  Teo Fabi
 14  Riccardo Patrese
 15  Nigel Mansell
 16  Stefan Johansson
 17  Gerhard Berger
 18  Jo Gartner
 19  Keke Rosberg
 20  Ayrton Senna
 21  Eddie Cheever
 22  Marc Surer
 23  Jonathan Palmer
 24  Martin Brundle
 25  Huub Rothengatter
 26  Jacques Laffite
 27  Stefan Bellof
 28  Piercarlo Ghinzani
 29  Francois Hesnault
 30  Manfred Winkelhock
 31  Johnny Cecotto
 32  Philippe Streiff
 33  Philippe Alliot
 34  Mike Thackwell
 35  Pierluigi Martini
```

3.2 Experimentation

I tested *initial_temperature* values as low as 0.1 and as high as 100
I tested *temperature_length* values as low as 1 and as high as 10000
I tested *cooling_ratio* values as low as 0.1 and as high as 0.99999
I tested *num_non_improve* values as low as 1 and as high as 100000

3.2.1 Observations

The value of 1 seems to be close to optimal for *initial_temperature*. Increasing or reducing it from 1 seems to decrease reliability. In both cases, Kemeny scores that are slightly worse than the optimal become more frequent. As expected, the amount of uphill moves made increases with *initial_temperature*.

Setting the *temperature_length* too high increases the uphill moves a lot, and decreases reliability, but has little impact on time. If temperature length and *num_non_improve* are increased together, then the time taken does increase substantially, but the results remain reliable. Setting *temperature_length* too low slightly increases the frequency of Kemeny scores that are significantly higher than usual (at least 10 higher).

a Setting *cooling_ratio* too high dramatically increases the frequency of Kemeny scores that are slightly worse than optimal. Setting it too low slightly increases the frequency of Kemeny scores that are significantly higher than usual. It appears to have little effect on runtime.

Overall, I have found that *num_non_improve* has the biggest effect on the Kemeny score reached. Lowering it appears to shift up the whole range of achievable Kemeny scores. The best and worst Kemeny scores all get worse. Increasing it enough seems to fix any problem caused by another parameter, but can increase runtime a lot.

With my final chosen parameters, my program (fairly consistently) achieves a Kemeny score of 62 with 0-2 uphill moves. Because of this, I expect that either the initial solution happens to be very close to the global optimum in the solution space, or (more likely) the global optimum is the local optimum for a

non-insignificant portion of the solution space. This is especially likely because there are multiple different ranking which yield a Kemeny score of 62 (shown in the screenshots).

I do still occasionally get higher-than-usual Kemeny scores. These are often accompanied by more uphill moves. The algorithm is not guaranteed to reach the local optimum of every peak it explores in the search space, however, the less peaks there are, the more improbable it becomes that the algorithm will not reach them. If there was only 1 local optimum for the whole solution space, the algorithm would have make 8000 moves without hitting it. This means it is extremely likely that there are multiple other local optima in the solution space, though they may not be as strongly sloping or large as the multiple global optima.

4 Binary Integer Program Formulation

4.1 Decision Variables

There are two decision variables X_{ij} and X_{ji} for each unordered pair of participants $\{i, j\}$, where $X_{ij}, X_{ji} \in \{0, 1\}$.

For each possible ranking, the value of X_{ij} is decided as follows:

$$X_{ij} = \begin{cases} 1, & \text{if participant } i \text{ is ranked better than participant } j \\ 0, & \text{otherwise} \end{cases}$$

The value for X_{ji} is decided the same way, just swap the is and the js.

4.2 Objective Function

For the objective function, we want to minimise the sum of all $a_{ij}X_{ji}$ terms, where there is one term for each value a_{ij} in the tournament matrix. Many of the values of a_{ij} will be zero, and can be ignored. The objective function for the example tournament in the coursework specification should look like this:

$$\text{Minimise } 5X_{AB} + 14X_{AC} + 2X_{CB} + 9X_{CD} + 3X_{DA}$$

4.3 Constraints

Each configuration of the decision variables can be thought of as a complete directed graph where there is a node for every participant, and the edges point towards the nodes with higher ranks.

To ensure a ranking is valid, we only have to show that it has no cycles. Because the graph is complete, we only have to check for cycles in each trio of nodes. This is stated in the coursework brief.

To ensure no cycles, we can say that for every unordered triplet of participants $\{i, j, k\}$, we must add two constraints:

$$X_{ij} + X_{jk} + X_{ki} < 3$$
$$X_{ij} + X_{jk} + X_{ki} > 0$$

These can be rewritten to use the same operator:

$$X_{ij} + X_{jk} + X_{ki} \leq 2$$
$$-X_{ij} - X_{jk} - X_{ki} \leq -1$$

In reality, we have two decision variables for each edge in our graph. Because of this we also need to ensure that their values are consistent with each other. If X_{ij} says that i is ranked above j, then X_{ij} should say the same thing. In other words, X_{ij} and X_{ji} must always have opposite values.

We must add the following constraint for each unordered pair of participants $\{i, j\}$:

$$X_{ij} + X_{ji} = 1$$

This can be reformulated to use the same operator as the previous constraints:

$$X_{ij} + X_{ji} \leq 1$$
$$-X_{ij} - X_{ji} \leq -1$$