Optimising for Entertainment in the Vote-Reveal Problem

Aric Fowler

May 2021

Abstract

In many elections or competitions, a set of voters will either rank a set of candidates from best to worst or will give scores to some of the candidates, with the winner then being the candidate that gets the highest total number of points. When it comes to revealing the result after all votes have been cast, some competitions proceed by having a roll-call of all the voters in which each announces their votes. This is often done for entertainment purposes (e.g., the Eurovision Song Contest), raising the question: Which ordering of voters should be chosen to maximise the entertainment value of the roll-call?

Contents

1 Introduction

The Vote-Reveal Problem (VRP) is difficult to solve due to the subjective concept of entertainment and the difficulty in finding an optimal solution among a factorially large set of feasible solutions. The aim of this project is to develop functions to measure the entertainment value of a given reveal order and to use optimisation techniques to find solutions that maximise these functions. The VRP is outlined in section 2 and an example instance is given. In section 3 the concept of entertainment value is discussed and six functions are presented to measure aspects of a solution's entertainment value. Section 4 covers the implementation of the VRP framework in addition to a visualisation tool to inspect solutions. Section 5 discusses single-objective combinatorial optimisation techniques and moves onto Tabu Search, its implementation, and a discussion of the solutions found. Section 6 covers relevant definitions and terminology for multi-objective combinatorial optimisation and presents Multi-Objective Tabu Search, its implementation, a discussion of the solution sets found, and a comparison to the results found using the original Tabu Search algorithm. Section 7 is an evaluation of the entertainment functions, the choice of algorithms, and the use of multi-objective combinatorial optimisation to approach the VRP. Section 8 is a brief reflection on learning and section 9 discusses possiblities for further work.

2 The Vote-Reveal Problem

The VRP is a combinatorial optimisation problem: the aim is to find the best (i.e. most entertaining) solution from a finite set of feasible solutions. Given a set of candidates $c_i \in C$ and a set of voters $v_j \in V$, the votes can be represented using an $i \times j$ matrix of weights

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1V} \\ w_{21} & w_{22} & \dots & w_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ w_{C1} & w_{C2} & \dots & w_{CV} \end{bmatrix}$$

Where:

w_{ij} is the vote weight given to candidate c_i by voter v_j

A feasible solution is any permutation of the voters, called a reveal order. Solutions can be represented using a permutation matrix where each decision variable d_{ij} indicates whether voter v_i reveals their vote at position j in the reveal. The constraints on these decision variables ensure that exactly one voter reveals their vote at any given point in the reveal and that every voter reveals their vote exactly once.

$$D = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1V} \\ d_{21} & d_{22} & \dots & d_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ d_{V1} & d_{V2} & \dots & d_{VV} \end{bmatrix} \qquad \begin{aligned} & d_{ij} = \{0,1\} \quad \forall (i,j) \\ & \sum_{x=1}^{V} d_{ix} = 1 \quad \forall i \\ & \sum_{x=1}^{V} d_{xj} = 1 \quad \forall j \end{aligned}$$

The objective function takes an assignment of D and W and returns a value E which represents the entertainment value of the solution according to some metric.

$$E = f_W(D)$$

Different functions can be used to measure the entertainment value; these will be covered in section 3.

2.1 Cumulative Values

In most instances of the VRP the audience does not only see the points revealed by each voter but also pays attention to the total points that each candidate has accumulated so far. These values infer positional information (e.g. which candidate is in the lead, which candidates are tied in points) and are usually the point of focus while watching a vote reveal take place.

Cumulative values can be calculated by iterating through each reveal from 1 to r and multiplying every voter's points by the decision variable indicating if those points are revealed.

$$\sigma(i,r) = \sum_{x=1}^{r}\sum_{y=1}^{V} W_{ix}D_{yx}$$

Where:

$\sigma(i,r)$	is candidate c_i's total points after the r^{th} reveal
W_{ix}	is the number of points given to c_i by voter v_x
D_{yx}	is 1 if voter v_x reveals their vote in position y, 0 otherwise

2.2 An Example Vote-Reveal Problem

Four voters $V = \{v_1, v_2, v_3, v_4\}$ are given a set $C = \{\text{bird}, \text{cat}, \text{dog}, \text{snake}\}$ of pets to choose from. Each voter ranks all pets from favourite (3 points) to least favourite (0 points). The votes are revealed in the order $\{v_1, v_2, v_3, v_4\}$.

$$W = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 3 & 2 & 3 \\ 0 & 2 & 3 & 2 \\ 3 & 0 & 0 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

	Reveal Stages			
bird	2	3	4	5
cat	1	4	6	9
dog	0	2	5	7
snake	3	3	3	3

Here, voter v_1 gave 2 points to bird, 1 to cat, 0 to dog, and 3 to snake. The winning candidate is cat with 9 points.

3 Measuring Entertainment Value

The VRP is especially difficult to solve due to the subjective nature of entertainment value. The entertainment of an audience can depend on any number of factors, including factors which rely on outside knowledge of the event[1]. Ely et al. [1] developed a model of "Suspense and Surprise" for revealing non-instrumental information to an audience. The model consists of a principal who reveals information over time and an agent who slowly learns the world state

[1]Meta-information is not considered in this project, but is a possibility for future work

using the revealed information. The agent has preferences over the possible outcomes and beliefs which determine their expected outcome depending on the revealed information. Suspense is defined as a period where the variance of the next period's belief is large while surprise is defined as a period where the current belief is very different to the previous period's belief. Bizzozero et al. [2] evaluated these factors using data from The Championships, Wimbledon, finding evidence to support their contribution towards entertainment value. Knobloch-Westerwick et al. [3] studied the effects of positive and negative affect (e.g., hope and despair) on suspense in American football spectators, finding that both affects contributed significantly to suspense regardless of if the preferred team was in the lead or not. Pawlowski et al. [4] found that respondents were more likely to watch a live sports match when the game is perceived to be suspenseful. They also noted that there is no evidence that being a fan of a participating team has an effect on the relationship between demand for sport and game uncertainty/suspense.

When choosing a function to measure entertainment value, the entertainer must consider the preferences of the audience members. For example: are they impartial viewers or supporters of specific candidates? The example of the Eurovision Song Contest is used in this project where there are a large number of candidates in each round, each representing their own country. Due to its nature as an international competition, a large number of audience members are supporters of their own country. However, unlike team sports where only two countries compete at a time, there is a large subset of countries present during each round. As a result of this, only a handful of candidates are perceived as having a chance of winning, and many audience members watch from an impartial standpoint where they are more interested in finding out who will win overall than in the outcome for their national candidate. With consideration for both supporters and impartial viewers, functions to measure entertainment value can be designed to focus on maximising surprise/suspense for either group.

3.1 The Lead Function

Algorithm 1 The Lead Function

```
procedure LEAD(W,R)
    $total \leftarrow 0$
    for stage $r$ in reveal do
        Get $C^* \leftarrow \sigma(i, r) \quad \forall i \in C$
        Sort $C^*$
        $d \leftarrow C^*[0] - C^*[1]$
        if $d < thres(W, R, r)$ then $total$ += 1
    return total
```

The Lead (LEAD) function considers the size of the lead that the first-place candidate has over second place. The aim is to limit the lead so that second place is consistently perceived as having a chance to catch up. This is achieved

Reveal Stages				
bird	2	3	4	5
cat	1	4	6	9
dog	0	2	5	7
snake	3	3	3	3
max 1	3	4	6	9
max 2	2	3	5	7
diff	1	1	1	2
thres	1.5	...		
$d < t$	T	T	T	F
output	3			

Table 1: The Lead function applied to reveal order $\{v1, v2, v3, v4\}$ in the example Vote-Reveal Problem

by comparing the cumulative values in first and second place at each stage of the reveal and calculating their difference. If the difference is below a threshold value $thres(W, R, r)$ then the score of the solution is increased by one. A perfect score ($f(W, R) = j$ where j is the number of voters) indicates that the lead remains below the threshold throughout the entire reveal. This may not be possible for some instances of the Vote-Reveal Problem. The threshold that was chosen for this project is the expected number of points gained in each round (i.e., the mean value over all elements of W). Table 1 gives an example evaluation of the Lead function.

3.2 The Hotseat Function

Algorithm 2 The Hotseat Function

```
procedure HOTSEAT($R, W$)
    $total \leftarrow 0$
    $i' \leftarrow 0$
    for stage $r$ in reveal do
        Get $C^* \leftarrow \sigma(i, r) \quad \forall i \in C$
        $C' \leftarrow \max(C^*)$
        $i \leftarrow$ candidate number for $C'$
        if $i \neq i'$ then
            $total$ += 1
            $i' \leftarrow i$
    return $total$
```

The Lead function considers uncertainty in the overall winner as static: It measures the difference in points at any given time. Another way to consider uncertainty is to look at how the first-place rank changes between candidates

Reveal Stages				
bird	2	3	4	5
cat	1	4	6	9
dog	0	2	5	7
snake	3	3	3	3
leader	snake	cat	cat	cat
change?	N/A	T	F	F
output	1			

Table 2: The Hotseat function applied to reveal order $\{v1, v2, v3, v4\}$ in the example Vote-Reveal Problem

throughout the reveal. While it may be entertaining for the second-place candidate to remain close behind first place, it could be seen as an improvement to the reveal if the two (or more) candidates swap around in their fight for first place. The Hotseat (HOT) function measures the number of times the first-place candidate changes throughout the reveal, with the aim to maximise this number.

3.3 The Unique Hotseat Function

Algorithm 3 The Unique Hotseat Function

```
procedure UniqueHotseat($R, W$)
    Initialise all flags as false
    for stage $r$ in reveal do
        Get $C^* \leftarrow \sigma(i, r) \quad \forall i \in C$
        $C' \leftarrow \max(C^*)$
        $i \leftarrow$ candidate number for $C'$
        set flag $i$ to true
    return $\sum f \quad \forall f \in$ flags
```

A possible downside of the Hotseat function is that, in reveal instances with more voters, it will reward reveal orders where the same two candidates fight back and forth for first place. This can be desirable in certain scenarios, however it may be useful to generate solutions where a high number of unique candidates take first place throughout the reveal. The Unique Hotseat (HOT-U) function addresses this by counting the number of unique candidates who have taken first place at any point in the reveal.

3.4 The Uncertainty Function

Previous functions have been tailored towards an impartial audience with the assumption that their interest lies only in the battle for first place through-

Reveal Stages				
bird	2	3	4	5
cat	1	4	6	9
dog	0	2	5	7
snake	3	3	3	3
leader	snake	cat	cat	cat
output	2			

Table 3: The Unique Hotseat function applied to reveal order $\{v1, v2, v3, v4\}$ in the example VRP

Algorithm 4 The Uncertainty Function

```
procedure UNCERTAINTY($R, W$)
    $total \leftarrow 0$
    for stage $r$ in reveal do
        Get $C^* \leftarrow \sigma(i, r) \quad \forall i \in C$
        Sort $C^*$
        for Cumulative value $C'$ in $C^*$ do
            $d \leftarrow$ MinDiff($C', C^*$)
            $total$ += $d$
    return $total$
```

out the reveal. If we instead assume that the audience consists of an equal number of supporters for each candidate, it is more appropriate to maximise uncertainty for all candidates. This can be achieved by minimising the gap between every candidate and their closest competitor throughout the reveal. The Uncertainty (UNC) function iterates through every stage of the reveal and measures the smallest difference in points between every candidate and every other candidate. The sum of these differences gives an overall score (representing the amount of uncertainty) which must be minimised. In implementation, the time complexity of this function can be reduced from $O(n^2)$ to $O(n \log n)$ by performing a quicksort on each row of cumulative values before measuring the differences. Sorting the cumulative values ensures that only the adjacent values in the array need to be checked instead of performing comparisons with every other value in the row.

3.4.1 Time-Biased Uncertainty

Consider a reveal where there are small point differences in the early stages but high differences later on. This solution may score very well according to the basic UNC function, however the audience may not be entertained by the reveal due to the relatively quick loss of uncertainty as the candidates' points start to spread out. The Time-Biased Uncertainty (UNC-T) function multiplies the differences by a time-based biased function, ensuring that solutions with

Reveal Stages				
bird	2	3	4	5
cat	1	4	6	9
dog	0	2	5	7
snake	3	3	3	3
Minimum Differences				
bird	1	0	1	2
cat	1	1	1	2
dog	1	1	1	2
snake	1	0	1	2
output	18			

Table 4: The Uncertainty function applied to reveal order $\{v1, v2, v3, v4\}$ in the example VRP

uncertainty towards the end of the reveal are preferred over solutions with high uncertainty at the start. The bias function chosen for this project is linear; the differences are multiplied by the current reveal number.

3.4.2 Candidate-Biased Uncertainty

The Candidate-Biased Uncertainty (UNC-C) function is an adaptation of the Uncertainty function to meet the assumption that impartial viewers are more interested in the eventual winner. This time, the domain of the bias function is the rank of each candidate instead of the reveal number. This focuses the search towards solutions with high uncertainty among the leading candidates. The candidate bias chosen for this project assigns a weight of 1 to the minimum differences of the first five candidates at each stage of the reveal, and 0 otherwise[2]. This was chosen to reflect the emphasis that is usually placed on the top three candidates as well as on any candidates that have a chance at getting onto the podium (i.e., fourth and fifth place).

4 Core Implementation

4.1 VRP Framework

An initial implementation of the VRP was developed before focusing on an optimisation method. The Java language was chosen for this task because it combines the benefit of fast, compiled code with an object-oriented paradigm that allows for the creation of modular code. Figure 1 gives a UML diagram of the classes used to implement the VRP framework. The `VRPDataReader` class

[2]Note that, given a bias function that gives a weight of 1 to the first-place candidate and 0 otherwise, UNC-C would reward a very similar set of solutions as LEAD.

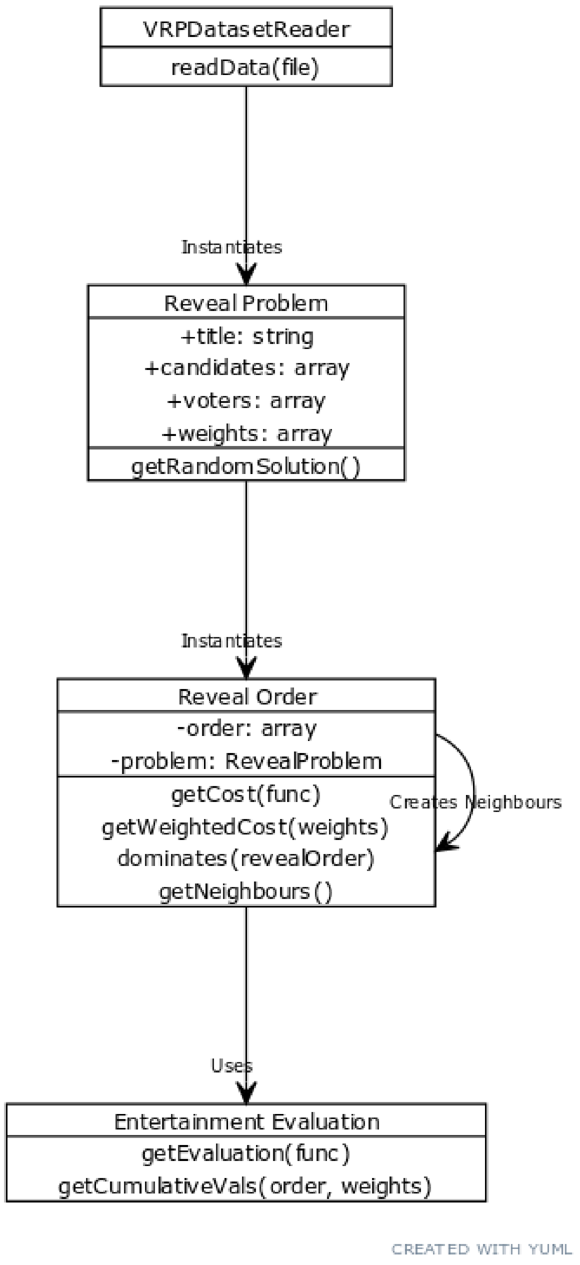

Figure 1: UML Diagram for the VRP framework

```
Example Vote Reveal      % Title
4                        % Number of candidates
bird                     % Candidate names
cat
dog
snake
4                        % Number of voters
v1                       % Voter names
v2
v3
v4
2, 1, 0, 3               % Weights (one line per voter,
1, 3, 2, 0               % votes in order of candidates)
1, 2, 3, 0
1, 3, 2, 0
```

Figure 2: Text template used to represent VRP instances

reads the problem instance from a file, so a template for notating VRP instances was created (listed in figure 2).

The `VRPDataReader` parses the template and creates an instance of `RevealProblem`. Random feasible solutions can be fetched from it with the method `getRandomSolution()`. This will be used by the optimisation algorithm to get a starting solution. The `RevealProblem` instance has public properties which are read-only so that any class can look up information about the VRP instance. The `RevealProblem` passes a self-pointer to any `RevealOrder` instances that it creates. Each `RevealOrder` can create neighbours of itself, each of which receives a copy of the pointer. This ensures that only one instance of the `RevealProblem` exists and prevents data duplication.

`RevealOrder` was designed with the `getCost()` and `getWeightedCost()` functions built into it so that any algorithm operating on instances of the class can easily evaluate them using the functions presented in section 3. Some of the objective functions are costly to evaluate, so a measure was put in place to prevent re-calculation of solution costs. Whenever `getCost()` or `getWeightedCost()` is called, the `RevealOrder` first checks if it has the relevant cost stored. If so, it uses the function name as a key to fetch the cost from a `HashMap` with a time complexity of $O(1)$. If the cost has not been calculated before, the `RevealOrder` calls `EntertainmentFunction` to calculate the cost and stores the cost in the `HashMap` before returning it.

Another improvement was made to the storage of cumulative values: all six of the entertainment functions require the use of cumulative values to evaluate a solution. Therefore, if a solution is measured using several functions, time can be saved by storing the cumulative values to prevent recalculation. Based on the assumption that every `RevealOrder` will be evaluated at least once, the cumulative values are calculated and stored privately during class construction so that they can be passed to `EntertainmentEvaluation` when required.

The `EntertainmentEvaluation` class contains static methods only. It is responsible for calculating the cumulative values for a given solution and evaluating a solution according a given entertainment function. The cumulative values are calculated by adding the current weight to the cumulative value from the previous row, reducing the time complexity from $O(n^3)$ to $O(n^2)$. The `getEvaluation()` function uses a switch on the function identifier. Since strings are not supported by java switches, the function is represented using a char. Objective evalution can be one of the slower parts of optimisation so execution speed was considered the main priority. All three variants of the uncertainty function use java's built-in Arrays.sort() method (which implements a merge sort with $O(n \log(n))$ average time complexity) to sort each row of cumulative values before iterating over them to check differences. This is faster than the $O(n^2)$ method where an iteration over the whole row is performed for each element.

4.2 Solution Visualisation Tool

An additional goal listed in the Initial Report was the development of a visualisation tool to inspect solutions. Java's Swing library was chosen to create the UI because of its widely available documentation. For the charts themselves, the open-source `JFreeChart` library was chosen for the same reason. The single-solution visualisation tool loads a solution and VRP instance and displays a bar chart of the candidates' points. The user can click "next" and "prev" to move forwards and backwards through the reveal order, with the current reveal number displayed next to the buttons.

Several highlighting modes were implemented to make it easier to spot certain features within a solution. These highlighting modes re-colour the bars from their default grey to the correct highlight colour. `JFreeChart`'s `BarChart` class uses a `BarRenderer` to colour each bar based on its row and column and was unable to highlight the bars as desired because of the limitation that all bars in the same row must be the same colour. This problem was overcome by creating a `CustomBarRenderer` which inherits from `JFreeChart`'s `BarRenderer` class and overrides the `getItemPaint()` function.

The highlighting information must be fetched from the solution visualiser, however, the `CustomBarRenderer` is not given a pointer to the visualiser when it is instantiated, leaving no way to access the visualiser instance. The solution visualiser was refactored to run statically from the `main()` method and the static `getColour()` method was added to the class so that it can be called by the `CustomBarRenderer` to fetch the colour for each bar. Highlighting rules are controlled by a set of checkboxes to allow any number of rules to be applied at a time. The rules include: (1) highlighting the current leader, (2) highlighting any candidates who have previously been in the lead, (3) highlighting tied candidates, and (4) highlighting candidates' positional changes. Examples of the highlighting modes are included in the appendix.

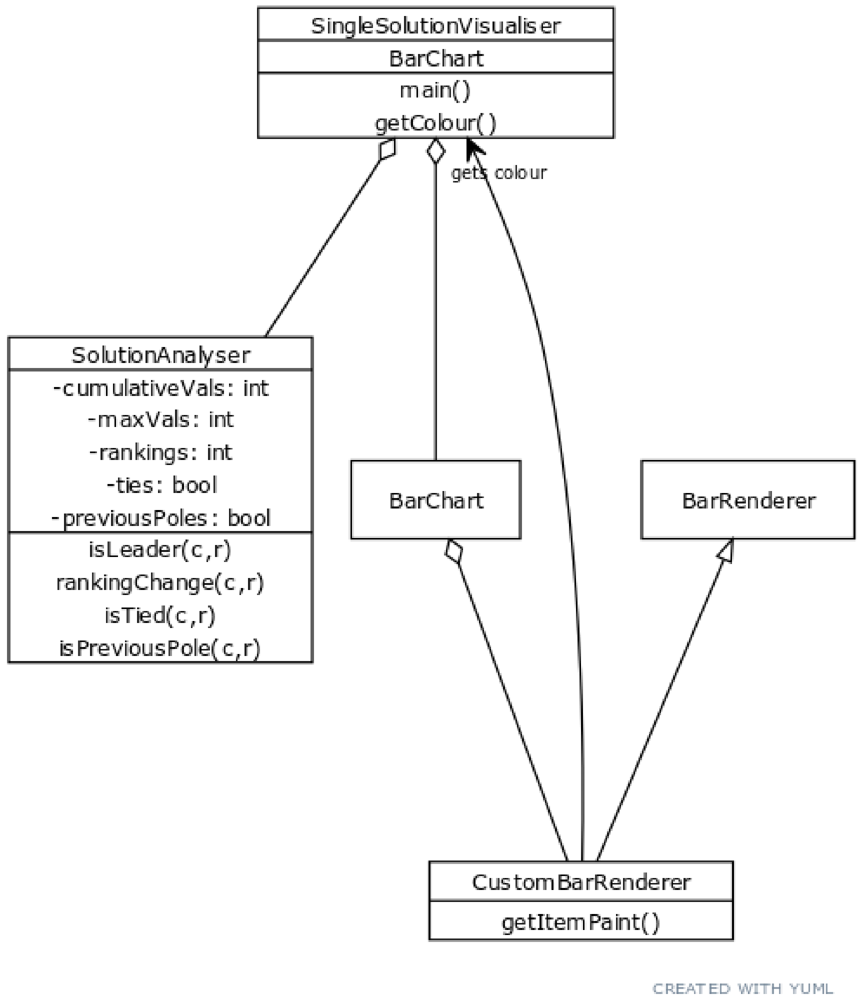

Figure 3: UML Diagram showing classes used to implement the single solution visualiser

The relevant stats that control the highlighting rules are calculated by the `SolutionAnalyser` class which takes a `RevealOrder` and iterates through the cumulative values to produce several tables with relevant values. It can then be queried by the visualiser using one of a set of functions (e.g., `isLeader()` or `isTied()`).

5 Single-Objective Optimisation

5.1 Background

It is not possible to use Gurobi for the Vote-Reveal Problem (as the initial report suggested) because Gurobi only supports linear and quadratic mixed-integer programming problems. Instead, it was decided to implement a combinatorial optimisation metaheuristic from scratch. A shortlist of single-solution search algorithms was produced using Gendreau's comparison of combinatorial optimisation metaheuristics [5]. The shortlist consists of (1) Greedy Randomised Adaptive Search Procedure, (2) Simulated Annealing, and (3) Tabu Search.

Greedy Randomised Adaptive Search Procedure (GRASP) uses a randomised greedy construction heuristic to produce initial solutions which are then improved using local search. After a fixed number of iterations, the best overall solution is returned. The construction heuristic builds up a partial solution by considering candidates to be added and creating a restricted candidate list consisting of the best candidates, from which a random candidate is selected and added to the partial solution.

Simulated Annealing (SA) is a randomised local search heuristic where a random neighbour of the current solution is selected and its cost is evaluated. If the neighbouring solution has a better cost (a downhill move), it is accepted as the new current solution. If the neighbouring solution has a worse cost (an uphill move), it has a probability of being accepted which depends on the current temperature. The temperature is gradually decreased according to some cooling schedule until some stopping criterion is met, at which point the search returns the best solution found so far.

Tabu Search (TS) is a deterministic local search heuristic where at each iteration the best neighbouring solution is selected, even if it is an uphill move. To avoid moving back towards previously visited local optima, TS uses a form of short-term memory called the tabu list. The tabu list can store either previously visited solutions or attributes of previous moves. When generating the neighbourhood of the current solution, any neighbours which are prohibited by the tabu list are not considered. The search continues until some stopping criterion is met, at which point it returns the best solution found so far.

All three metaheuristics are feasible for use in the Vote-Reveal Problem. GRASP was considered out-of-scope for this project due to its solution construction approach; the entertainment functions all use cumulative values and as such it would be very difficult to meaningfully evaluate candidates for partial solutions when future cumulative values cannot be predicted. The decision was ultimately between TS and SA, with TS being selected because of its deterministic approach and the use of short-term memory to inform decisions.

5.2 Tabu Search

Algorithm 5 Tabu Search Algorithm as described in *A user's guide to tabu search* [6]

```
procedure TabuSearch(s)
    $s^* \leftarrow s$
    $k \leftarrow 1$
    while stopping criterion not met do
        $k \mathrel{+}= 1$
        Generate $V^* \subseteq N(s^*, k)$
        Choose the best $s'$ in $V*$
        $s \leftarrow s'$
        if $f(s') < f(s^*)$ then $s* \leftarrow s'$
```

The two important components of Tabu Search are the neighbourhood function and the tabu list, which work together to generate $N(s^*, k)$. In permutation problems it is common to use pairwise swaps of elements to define a neighbourhood and to keep track of these pairs in the tabu list (instead of recent solutions) [7]. The size of this neighbourhood has an $O(n!)$ relationship with the number of voters, making the evaluation and sorting of the neighbourhood slow for larger VRP instances. To improve execution speed, a smaller neighbourhood can be used where each neighbour is generated by swapping adjacent voters in the current reveal order. The adjacency requirement for swapped elements reduces the size complexity of the neighbourhood to $O(n)$. Both the reduced and full neighbourhood functions will be used and compared. Various stopping criteria can be used; the criterion used in this project checks if the number of iterations since the best solution was last updated has exceeded a set limit.

5.3 Implementation

Using the existing VRP framework, the implementation of TS remained very true to the pseudocode in algorithm 5. The tabu list was implemented in its own class with add() and contains() functions. Over time the TS algorithm was re-worked to optimise for speed, however the algorithm was very fast to begin with and there was little difference in execution time after the changes were made.

The first optimisation was to change the implementation of the tabu list. The original `TabuList` class kept a list of actions and performed a linear search when the `contains()` method was called. It was able to add new actions with complexity $O(1)$ and check if an action was contained in the list with worst-case complexity of $O(t)$ (where t is the tabu list length). The `TabuListMap` class

	Eurovision Final 2018				Eurovision Final 2019			
	Random		TS		Random		TS	
	Mean	Best	R	F	Mean	Best	R	F
LEAD	3.35	18	16	21	6.19	22	21	25
HOT	2.92	17	14	20	4.59	20	20	23
HOT-U	2.64	8	7	8	3.25	9	8	11
UNC	2740	1752	1697	1431	2495	1683	1550	1321
UNC-T	44816	29620	29096	25295	40713	27942	27084	24108
UNC-C	1552	638	544	450	927	336	291	201

Table 5: Best scores achieved in 50 attempts with Tabu Search using reduced (R) and full (F) neighbourhood functions compared to the mean and best scores using 1 million random solutions.

uses a `HashMap` to store actions, allowing the contains method to be executed in $O(1)$ time complexity. The tradeoff is that each action contained in the map needs to individually store a value indicating how long they've been in the list, which increases the complexity of adding elements to $O(t)$ because each element in the map needs to be updated. Since the contains function is used more often than the add function, this should theoretically improve performance however the difference in the overall execution time of the TS algorithm was negligible.

The second optimisation was a change in the generation of neighbours. The first TS implementation would iterate over all possible neighbours given by the `getNeighbours()` function to find the first neighbour that is allowed by the tabu list. This required the costly evaluation and sorting of every neighbour. Time spent on these costly operations can be reduced by discarding tabu moves before generating the neighbours. The process was refactored so that the TS algorithm would generate the list of actions itself and discard any actions which are prohibited by the tabu list. The reduced list of "legal actions" is then passed to the current solution, which evaluates and sorts each neighbour represented by an action. The new algorithm was run using a non improve limit of 1000 to ensure sufficient iterations and a list length of 300 to ensure that a significant number of moves were considered tabu at any time. Despite the considerable reduction in the number of solutions being evaluated with the new system, the time to perform 1000 iterations remained unchanged (about 4.32 seconds).

5.4 Results

TS was tested using example data from the Eurovision Song Contest Database[3]. Two instances of the Vote-Reveal Problem were constructed: one from the 2018 grand final and the other from the 2019 grand final (these datasets are listed in tables 7 and 8 in the appendix). Benchmarks for each of the six evaluation functions were generated for both VRP instances by evaluating one million

[3]Available to the public at *https://eschome.net/*, accessed 2021

randomly generated feasible solutions. The mean score and best overall score from the samples are given in table 5 alongside the best solutions found with TS.

The tabu list size and non-improve limit were decided based on experimentation. Glover's guidance for choosing the tabu list size is to watch for "the occurrence of cycling when the size is too small and the deterioration in solution quality when the size is too large" [6]. In both VRP instances used, the reduced neighbourhood function could produce a maximum of 25 neighbours and the full neighbourhood function 325. Even with the tabu list length very high (20 for reduced and 300 for full), the best solution was found very quickly (i.e., within 20 iterations or so) and the search was unable to improve on this solution. This could be an indicator of cycling (although the very high tabu list length would presumably combat this) or it could perhaps indicate very wide and tall barriers between local optima. As such, the non-improvement limit was brought down to 100 and later to 50 with the list length being set to 18 for reduced neighbourhood and 250 for full neighbourhood.

The reduced neighbourhood function performed very poorly compared to full neighbourhood, although it was significantly faster. For LEAD, HOT, and HOT-U, the reduced neighbourhood was unable to produce a better score in 50 attempts than the best score found through random sampling. It was, however, able to beat the randomly sampled results in all three variants of the uncertainty function. The full neighbourhood search found the best solutions for every objective, beating the reduced neighbourhood and random sampling results by large margins in some cases.

6 Multi-Objective Optimisation

When trying to model entertainment value, single-objective optimisation can be limiting. In reality, the most entertaining solution may be entertaining due to a variety of factors that all contribute in different ways. A solution that scores well on multiple entertainment measures may be preferred to a solution that is optimal on a single measure but scores poorly on all others. After the completion of the Tabu Search algorithm and solution visualisation tool, it was decided to extend the project to cover multi-objective optimisation and to consider its effectiveness in the VRP. In addition to implementing a multi-objective optimisation algorithm, a second visualisation tool was developed to compare the performance of solutions across several objectives.

6.1 Background

It is possible to combine multiple functions into one objective that produces a weighted output of their combined scores, however the loss of dimensionality creates difficulty in optimisation and obscures each objective's individual scores from the search algorithm. Therefore it is more effective to use multi-objective optimisation techniques. The explanation below is a brief summary of information given by Hansen [8].

Multi-Objective Combinatorial Optimisation (MOCO) problems take the following general form:

$$\begin{aligned} \text{Maximise} \quad & f(x) \\ \text{Subject to} \quad & x \in S \\ \text{Where} \quad & f(x) = \{f^1(x), f^2(x), \ldots, f^n(x)\} \\ \text{and} \quad & S \text{ is the set of feasible solutions} \end{aligned}$$

The word "maximise" here is not strictly accurate because it is usually impossible to find a single solution that has the optimal value for each objective. For any solution x_n, its cost $f(x_n)$ is a vector and can be considered a point in cost-space (the word "point" is used to refer to a solution's cost evaluation). Solution x_1 is "superior" to x_2 if the point $f(x_1)$ dominates $f(x_2)$, which occurs when $f^i(x_1) \geq f^i(x_2)$ for all objectives and $f^i(x_1) > f^i(x_2)$ for at least one objective. Therefore, domination indicates that a point is *strictly better than* another point. A Non-Dominated (ND) Set consists of solutions where no solution dominates any other solution in the set (therefore no single solution is strictly better than any other solution). By definition, the ND set is pareto optimal: increasing one value by moving to a different item in the set will always result in a decrease in value on another axis. The optimal ND set is a set where there exist no feasible solutions which dominate any solution in the set. The act of finding the optimal ND set is very difficult and therefore the aim of MOCO is to obtain a reasonable estimate of the ND set. The example VRP only has 24 feasible solutions, meaning that the optimal ND set can be found through brute force. This set (shown in figure 4) contains four solutions, each of which achieves the optimal value for all six objective functions.

6.2 Multi-Objective Tabu Search

Multi-Objective Tabu Search (MOTS) was proposed in 1997 [8]. The procedure uses a population of current solutions, each with its own tabu list, which are iterated over and optimised towards the Non-Dominated frontier. For each solution, an optimisation direction is chosen which assigns a weight to each objective. The distance between two solutions can be measured as the distance between their points; a function of the distance is used to give nearby points a greater influence on the optimisation direction in order to spread the search across the pareto frontier. Neighbours are sorted by their weighted score (calculated as the sum of each normalised objective cost multiplied by some weight) so that the neighbour that maximises the score in the desired direction is selected first. When calculating the optimisation direction, population points which are dominated by the current point are ignored.

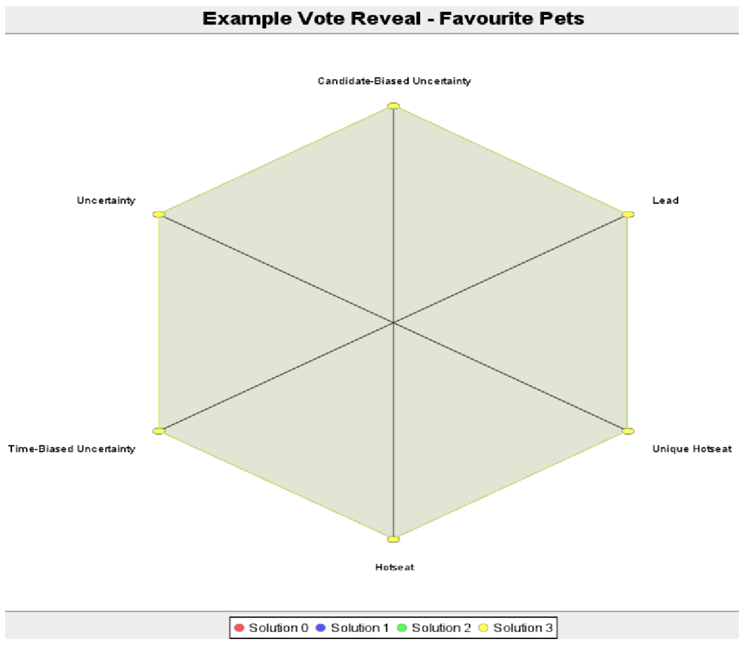

Figure 4: Optimal ND set for the example VRP instance

Algorithm 6 Multi-Objective Tabu Search

```
procedure MOTS
    Initialise $X$ as a set of random feasible solutions
    Initialise ND $\leftarrow \emptyset$
    $k \leftarrow 0$
    while stopping criterion not met do
        for solution $x_i \in X$ do
            $\lambda \leftarrow 0$
            for solution $x_j \in X$ where $f(x_i)$ is non-dominated by $f(x_j)$ and $f(x_i) \neq f(x_j)$ do
                $w \leftarrow g(d(f(x_i), f(x_j), \pi))$
                for objective $k$ where $f^k(x_i) > f^k(x_j)$ do
                    $\lambda^k += \pi^k w$
            if $\lambda = 0$ then set $\lambda$ to random weights
            normalise $\lambda$
            find solution $y_i \in N(x_i, k)$ which maximises $\lambda * f(y_i)$
            $x_i \leftarrow y_i$
            if $f(y_i)$ non-dominated by all points in ND then
                ND $+= y_i$
                Remove solutions in ND that are dominated by $f(y_i)$
            $k += 1$
    return ND
```

6.3 Implementation

6.3.1 MOTS Algorithm

The main MOTS algorithm was implemented as seen in algorithm 6. The calculation of weights is the same, with random weights being assigned by choosing a random float from 0 to 1 for each element and then normalising the vector so that it sums to 1. This is sufficient for the purpose of choosing a random vector since no specific probability distribution is required. Domination was handled by implementing a `dominates()` function in `Revealorder`. This allows a solution to compare itself to another solution and return true or false accordingly. Additional neighbourhood functions and evaluation functions were added to `RevealOrder` to support the use of a weights vector across several objectives. The range equalisation factors could have been hard-coded for each VRP instance and objective, however this will not extend to new VRP instances or new objective functions. Therefore, the equalisation factors are evaluated in the initialisation stage using 100,000 random solutions[4]. Distances between points are calculated using Manhattan distance as recommended by Hansen.

6.3.2 Solution Comparison Tool

The solution comparison tool reads a list of solutions from a CSV file and creates the appropriate `RevealProblem` instance. It then displays each solution as a coloured polygon on a spider chart. The `SpiderChart` class evaluates the costs for each pairing of solution and objective and scales them to the range $[0, 1]$. This must be done because `JFreeChart`'s spider chart does not normalise the axes and does not support negative values[5]. Examples of the multi-solution visualisation tool can be seen in figures 5 and 6.

6.4 Results

MOTS was tested using the same VRP instances as TS with the same settings (list length 20, non-improve 50) using the full neighbourhood function. A sample size of 50 was used to match the 50 restarts given to TS. The ND set for the 2018 data contained 50 solutions and the ND set for the 2019 data contained 122 solutions. The sets were both too large to meaningfully compare, so the sets were trimmed by hand in an excel spreadsheet. Solutions that did not have the optimal value on at least one measure were removed, and if any solutions existed which had the optimal value on multiple measures, solutions which only had a single optimal value on one of said measures were removed. The sets were reduced to 4 solutions and 6 solutions respectively.

Table 6 compares the best solution scores found for each function using TS and MOTS. MOTS achieved very similar results, even beating TS on the two

[4]Perhaps a cleaner system would check a file for equalisation factors and, if they are not found, generate the factors and write them to the file for future reference.

[5]Values calculated for the uncertainty functions are returned as negative values so that every function can be treated as ”maximising” by the optimisation algorithm

Instance	Eurovision Final 2018		Eurovision Final 2019	
Algorithm	TS	MOTS	TS	MOTS
LEAD	21	21	25	25
HOT	20	21	23	24
HOT-U	8	9	11	12
UNC	1431	1639	1321	1541
UNC-T	25295	28246	24108	25877
UNC-C	450	511	201	239

Table 6: Comparison of best scores achieved with TS (50 attempts) and MOTS (1 attempt, sample size 50).

hotseat functions. MOTS did not perform as well as TS on the uncertainty functions, falling slightly behind on all three with both instances. Overall the best values found with MOTS are about the same as TS.

There is the possibility that solutions found with TS could coincidentally perform well on other objectives. This depends on similarities in the way each objective is evaluated as well as a bit of luck. To observe these links between solutions, ND sets for both VRP instances were created using the optimal solutions produced by TS (see figures 11 and 12 in the appendix). In both cases, solutions optimised for Uncertainty performed very well in Time-Biased Uncertainty and vice versa (in fact, in the 2018 set the UNC-T solution outperforms the UNC solution on its own metric). This is not the case for Candidate-Biased Uncertainty, with both of the UNC-C solutions performing poorly on UNC and UNC-T. This difference is most likely due to the difference in bias function used, with the linear bias producing similar solutions to unbiased while the cutoff bias ignores part of the dataset. In both sets, the Hotseat and Unique Hotseat functions do not perform well on the other objective, indicating the lack of overlap there. The Hotseat solution in both sets performs very well on the Lead function, indicating an overlap where - intuitively - reveal orders with a high number of changes at pole position will likely have a very small gap between the top two candidates at any time. The Unique Hotseat solutions in both sets consistently perform poorly in all other metrics, indicating that the HOT-U function has very little overlap with other objectives. Coincidentally, the Uncertainty solution in the 2018 dataset matched the optimal HOT-U value. While high-scoring Uncertainty solutions have the possibility of scoring well on the Hotseat functions (because they group the candidates together as much as possible before the candidates inevitably drift apart in points), there is no explicit incentive to have a high HOT score.

The ND sets produced by MOTS show a significant improvement in the scores of solutions across all objectives. For example, solution 0 in the 2019 set holds the optimal value for the LEAD and UNC-C objectives and scores well in three of the remaining objectives. Solution 0 of the 2018 set holds three optima and performs consistently well in the other three objectives. Once again, the

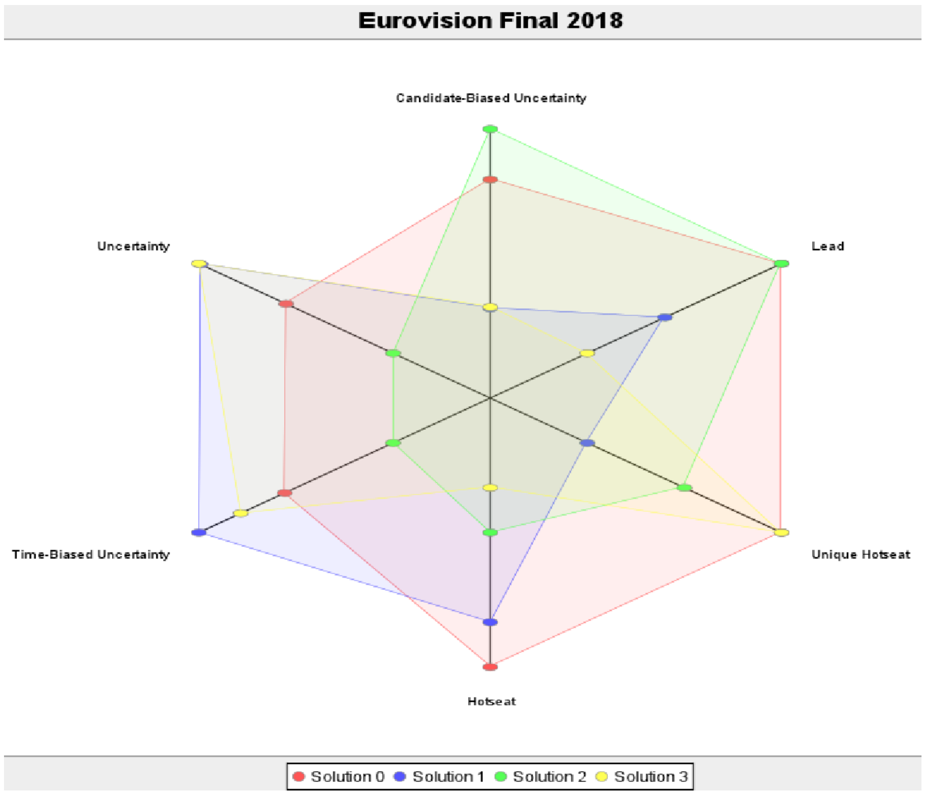

Figure 5: ND set for the 2018 eurovision final

best solutions in UNC and UNC-T tend to perform well in the other function, with solution 1 from the 2018 set and solution 4 from the 2019 set holding optimum or near-optimum values in both objectives.

7 Evaluation

7.1 Models for Entertainment

Optimised solutions for each objective function successfully captured the intended results, although some with more success than others. It is very easy to observe the changes between leaders when looking at high-scoring HOT and HOT-U solutions. Both of these functions achieve their intended outcome very well. The LEAD and UNC-C functions were successful, with high-scoring solutions showing very small gaps in points between the leading candidates. In high-scoring UNC-C solutions it is often very clear that the top 5 candidates are in close competition. The success of the UNC-C function may be due to the weighting of the bias which ignores any candidates not in the top five. Solutions

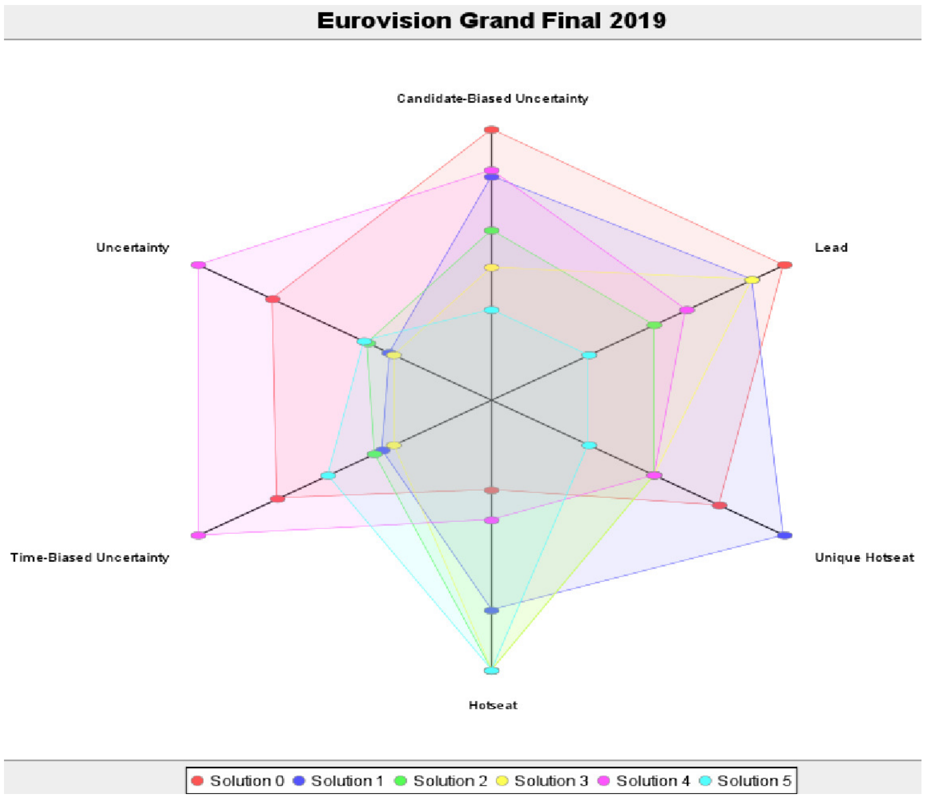

Figure 6: ND set for the 2019 eurovision final

optimised for LEAD are sometimes underwhelming; this may be due to the focus on the top two candidates being too narrow, reducing the payoff and making the function very dependant on the reveal instance (e.g., in the 2018 instance the lead function performs poorly because of the dataset - Israel had a very popular candidate and won by a large margin). Solutions optimised for UNC and UNC-T do well at creating small gaps in points - often managing to create many ties in the rankings throughout the reveal. The presence of so many ties may be helped by the new eurovision scoring system which combines national votes and panel votes, since it is often the case that slight differences in the preference order produce ties in the number of votes assigned by the country as a whole. This can be observed in many solutions optimised for UNC where the very first vote that is revealed places the majority of point-receiving candidates into a tie with at least one other candidate. Many solutions optimised for UNC have the majority of small point differences in the early stages of the reveal, likely because all candidates start at zero points and it is easier to minimise point differences towards the start before the candidates inevitably spread out to reach their final point counts. Despite the emphasis that UNC-T places on the later stages of the reveal, there were no significant noticeable differences between these solutions and solutions optimised for UNC. To achieve more noticeable results It may be necessary to use a more aggressive bias function such as the one used for UNC-C.

7.2 Tabu Search

Both TS and MOTS outperformed random sampling significantly but it is not clear if the TS metaheuristic is ideal for the VRP. Even when using very high tabu list sizes, the algorithm consistently found the optimal solution within 20 iterations and would return after the non-improve limit. This is an indicator that the search may be returning the first local optima that it finds and is unable to escape the pit to move towards new local optima. It may be the case that additional concepts such as intensification and diversification need to be applied to TS to allow it to move between optima, or perhaps a different metaheuristic would have unique advantages that allow it to overcome this problem (e.g., GRASP's solution construction approach). It could equally be the case that the solution space has exceedingly high amplitude in cost evaluations, therefore different approaches may be required such as population-based algorithms or the use of larger neighbourhood functions.

7.3 Application of MOCO to the VRP

While both algorithms achieved simlar optima for each objective function, MOTS had the significant advantage of producing solutions that maximise several functions (and often meet the optimum value in several functions). Multi-Objective Combinatorial Optimisation is very appropriate for VRP because it addresses the difficulty in defining entertainment value in any single cost function by allowing entertainment value to be modelled as multi-faceted, therefore generating solutions that try to meet a set of goals. Another advantage is that the resulting Non-Dominated set can feasibly be reduced by the entertainer to a set of options to choose from. This gives the entertainer the advantage of being able to view a small and manageable set of solutions in order to choose one that meets their requirements. The results generated with MOCO can be seen as a set of recommendations instead of a final answer, placing the power of selecting a final solution back in the hands of a human who can use intuition to choose a solution based on factors outside of those modelled by the set of objective functions.

8 Reflection on Learning

Although I learned about Tabu Search in CM3109 Combinatorial Optimisation, I did not have the opportunity to fully familiarise myself with it until this project where I developed and tested my own implementation of the algorithm from scratch. This project has allowed me to further explore the field of combinatorial optimisation while working on a previously unseen problem. The focus on measuring entertainment value has allowed me to come up with my own ideas to evaluate solutions and I have learned how to incentivise certain features of a solution by designing and implementing functions that model my ideas. A notable learning point involving the objective functions was the use of biases to create variants of the Uncertainty function. The idea of using a higher-order function was very interesting to me, and I think that it can be very useful to model entertainment value because of the flexibility of applying any bias function to the general Uncertainty function in order to apply different preferences. In the future, it could be interesting to consider more higher-order functions and to perform tests on various configurations to see which functions produce desirable results.

The main learning area of this project came from my exploration of Multi-Objective Combinatorial Optimisation. MOCO was not considered until part-way into the project when I had completed the TS algorithm and visualisation tool. At first I had developed ideas for my own algorithm to simultaneously optimise several objectives, however after doing some research I came across the field of multi-objective optimisation and read about important concepts such as dominance and pareto optimality. I realised that my algorithm idea was limited compared to the approaches used by others, however I thoroughly enjoyed exploring the field of multi-objective optimisation and will likely continue to read about its applications and the ways in which single-objective metaheuristics can be adapted to optimise over multiple objectives.

9 Further Work

Further work on the VRP could cover a wide range of new approaches. Any number of new objective functions can be devised to model particular desirable features of the reveal order. This includes the consideration of meta-factors from the audience's perspective: Has the audience already seen the performance of each candidate? How have the candidates performed in past competitions? Are there any known biases among the voters? Other insights into definitions of entertainment for VRP could be gained from interviewing audience members and there may be some use in presenting solutions to volunteer audiences in order to measure how well the model being used reflects the actual entertainment of the audience. Different optimisation techniques may yield better results for the same set of objectives; there are variants of the Tabu Search procedure not explored within this project which could perform better and be more suitable to the Vote-Reveal Problem. It could also be useful to develop an algorithm to trim the ND set to produce a smaller set of recommendations that are likely to be desirable to the entertainer.

10 Acronyms

GRASP Greedy Randomised Adaptive Search Procedure. 14, 24

HOT Hotseat. 1, 6, 7, 16, 17, 21, 22

HOT-U Unique Hotseat. 1, 7, 8, 16, 17, 21, 22

LEAD Lead. 1, 5, 6, 9, 16, 17, 21–23

MOCO Multi-Objective Combinatorial Optimisation. 2, 18, 24, 25

MOTS Multi-Objective Tabu Search. 2, 18–21, 24

ND Non-Dominated. 18–22, 24, 25

SA Simulated Annealing. 14, 15

TS Tabu Search. 2, 14–17, 20, 21, 24, 25, 29, 30

UNC Uncertainty. 1, 7–9, 16, 21–25

UNC-C Candidate-Biased Uncertainty. 9, 16, 21, 22, 24

UNC-T Time-Biased Uncertainty. 8, 16, 21–23

VRP Vote-Reveal Problem. 1–4, 6–12, 14–21, 24, 25

11 Appendix

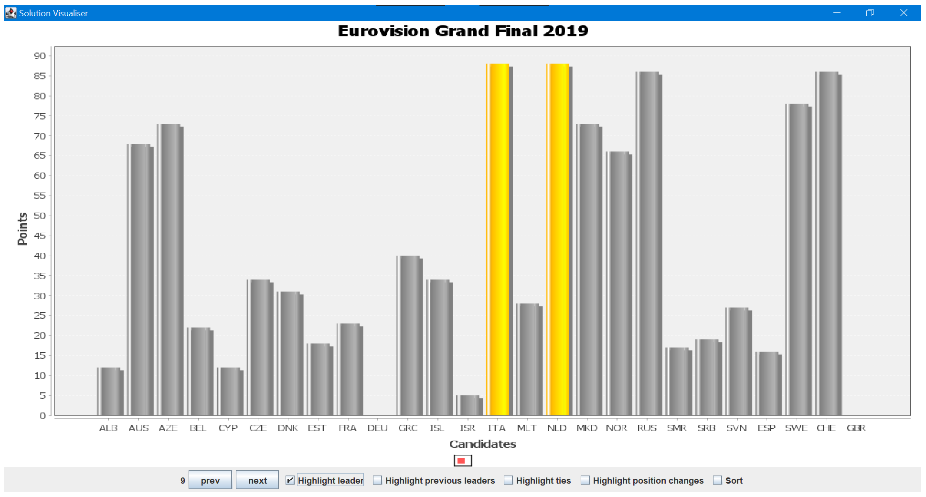
Solution Visualiser
Eurovision Grand Final 2019
Points
Candidates
ALB AUS AZE BEL CYP CZE DNK EST FRA DEU GRC ISL ISR ITA MLT NLD MKD NOR RUS SMR SRB SVN ESP SWE CHE GBR
9 prev next
Highlight leader
Highlight previous leaders
Highlight ties
Highlight position changes
Sort

Figure 7: Visualisation with leader(s) highlighted in yellow

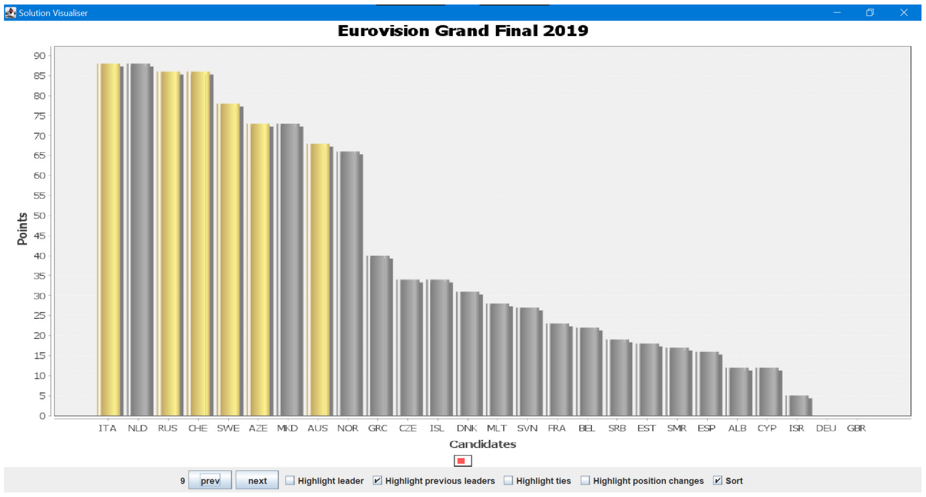
Solution Visualiser
Eurovision Grand Final 2019
Points
Candidates
ITA NLD RUS CHE SWE AZE MKD AUS NOR GRC CZE ISL DNK MLT SVN FRA BEL SRB EST SMR ESP ALB CYP ISR DEU GBR
9 prev next
Highlight leader
Highlight previous leaders
Highlight ties
Highlight position changes
Sort

Figure 8: Visualisation with previous leaders highlighted in gold

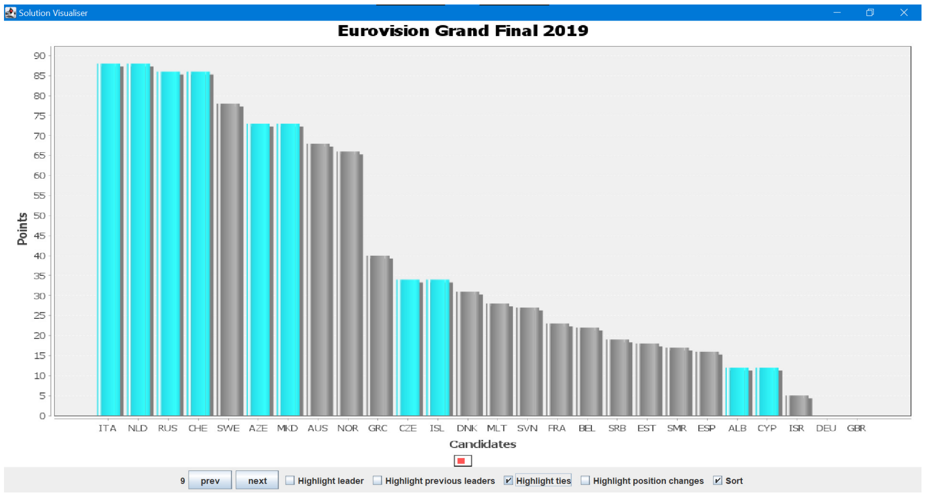
Solution Visualiser
Eurovision Grand Final 2019
Points
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
ITA NLD RUS CHE SWE AZE MKD AUS NOR GRC CZE ISL DNK MLT SVN FRA BEL SRB EST SMR ESP ALB CYP ISR DEU GBR
Candidates
9 prev next
Highlight leader
Highlight previous leaders
Highlight ties
Highlight position changes
Sort

Figure 9: Visualisation with ties highlighted in blue

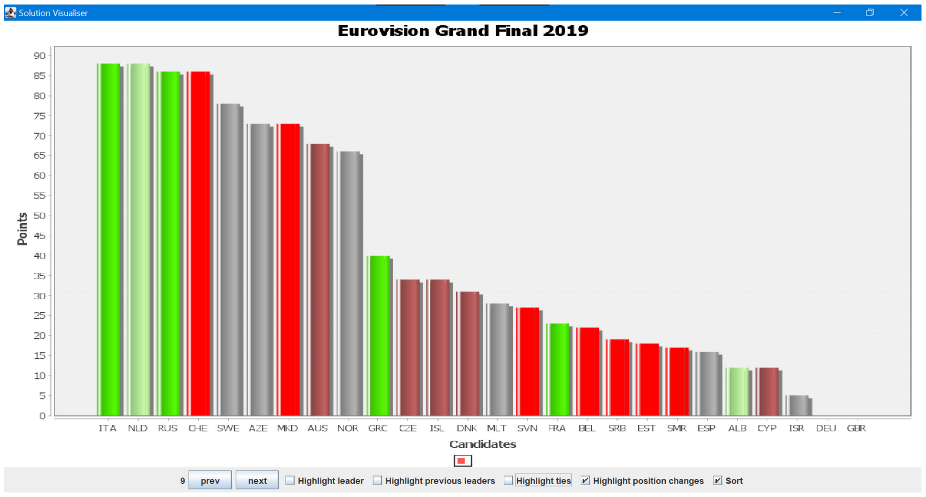
Solution Visualiser
Eurovision Grand Final 2019
Points
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
ITA NLD RUS CHE SWE AZE MKD AUS NOR GRC CZE ISL DNK MLT SVN FRA BEL SRB EST SMR ESP ALB CYP ISR DEU GBR
Candidates
9 prev next
Highlight leader
Highlight previous leaders
Highlight ties
Highlight position changes
Sort

Figure 10: Visualisation with candidate rank changes highlighted in green and red

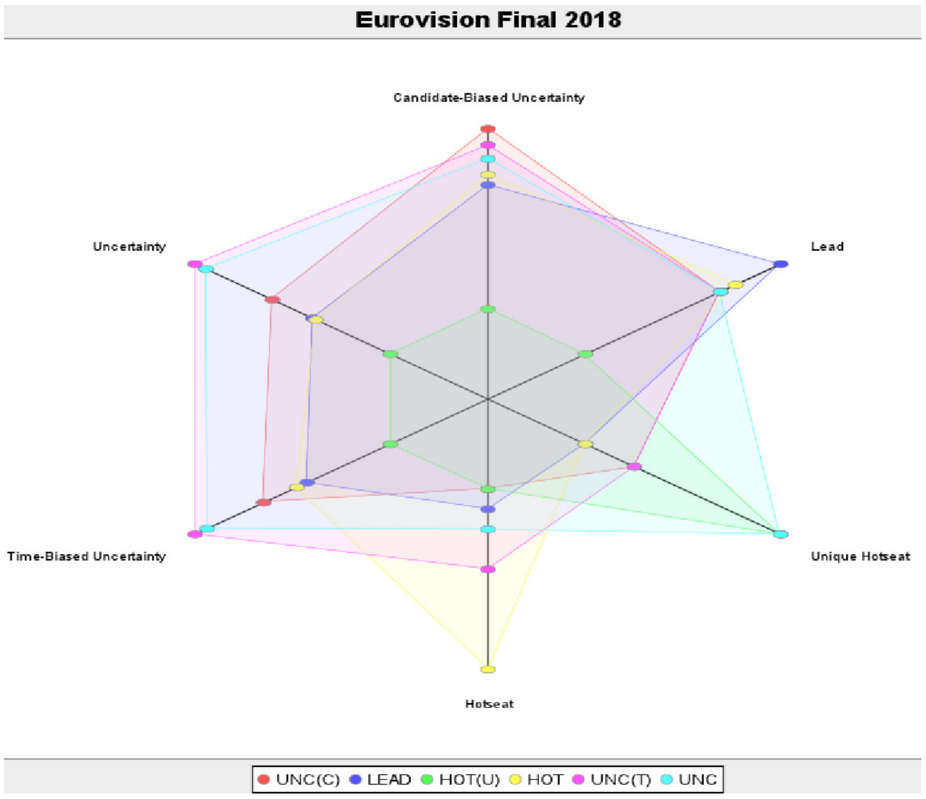

Figure 11: Optimal solutions generated by TS for 2018 dataset

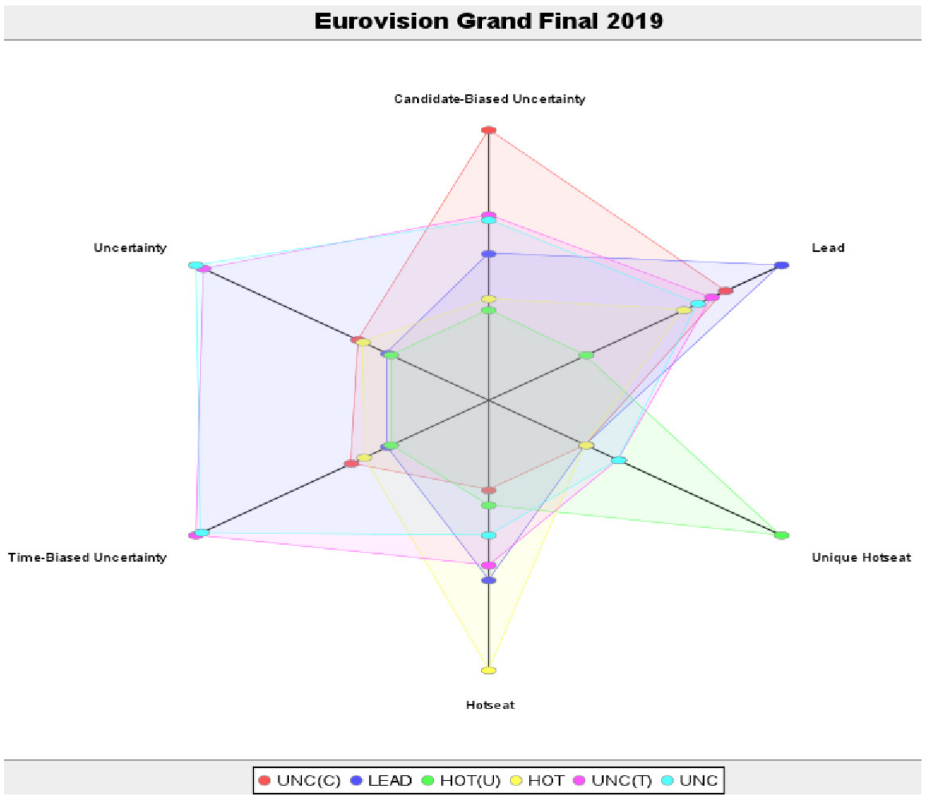

Figure 12: Optimal solutions generated by TS for 2019 dataset

	Candidates																									
Voters	ALB	AUS	AUT	BGR	CYP	CZE	DNK	EST	FIN	FRA	DEU	HUN	IRL	ISR	ITA	LTU	MDA	NLD	NOR	PRT	SRB	SVN	ESP	SWE	UKR	GBR
ALB	0	0	2	14	20	1	0	4	0	12	14	0	7	6	24	0	2	0	0	0	3	0	0	4	0	3
AUS	0	0	5	0	7	4	10	8	4	0	12	0	12	18	0	3	7	0	1	0	0	0	7	12	0	6
AUT	9	0	0	6	1	15	5	2	0	0	16	3	8	19	10	0	0	1	0	0	8	5	0	8	0	0
BGR	7	0	16	0	15	11	0	2	0	5	6	5	1	14	6	10	8	0	0	0	7	0	0	2	1	0
CYP	6	0	2	12	0	13	0	5	0	4	3	4	0	10	15	6	10	0	5	0	0	0	7	12	2	0
CZE	6	0	5	5	8	0	7	5	3	0	3	2	14	22	2	1	6	0	0	0	0	7	0	8	12	0
DNK	0	12	18	1	6	6	0	7	0	2	24	0	4	3	0	0	0	5	8	0	0	0	6	11	0	3
EST	0	0	18	7	12	5	8	0	12	0	6	3	0	0	7	22	1	1	6	3	0	0	0	5	0	0
FIN	0	0	10	10	7	5	13	12	0	9	1	8	2	19	10	0	0	2	0	0	0	0	0	8	0	0
FRA	0	10	7	0	3	4	2	7	0	0	8	0	1	24	10	0	6	6	0	8	1	2	5	5	4	3
DEU	0	7	15	0	9	8	3	2	0	0	0	2	15	11	12	8	0	5	0	0	0	0	6	12	0	1
HUN	10	7	11	0	7	8	24	0	0	0	1	0	3	16	6	5	2	8	7	0	0	0	0	1	0	0
IRL	2	0	5	10	17	7	2	7	0	4	16	0	0	13	0	12	1	3	0	6	0	0	1	0	0	10
ISR	0	7	13	1	2	12	6	8	6	6	8	3	0	0	5	0	10	0	0	0	0	4	0	10	7	8
ITA	12	0	7	0	8	0	12	10	0	0	13	2	5	9	0	0	10	0	12	0	1	0	0	1	8	6
LTU	0	0	15	0	7	8	10	12	0	15	7	2	4	7	7	0	0	3	0	7	0	0	0	12	0	0
MDA	0	7	3	11	13	6	0	13	0	0	8	2	0	22	8	0	0	1	7	0	0	0	0	0	15	0
NLD	0	0	13	0	11	6	8	4	0	0	24	2	4	15	7	7	0	0	4	2	0	1	0	8	0	0
NOR	0	6	16	0	7	4	10	0	0	8	18	0	1	7	0	15	0	9	0	0	0	0	2	13	0	0
PRT	10	0	8	7	5	0	2	19	0	5	8	0	3	2	14	6	6	0	0	0	0	3	14	0	4	0
SRB	1	0	4	2	10	5	4	3	0	0	10	12	0	9	14	0	6	7	9	0	0	8	0	12	0	0
SVN	5	0	12	0	14	11	7	5	0	2	4	3	0	1	10	0	6	7	5	0	12	0	0	12	0	0
ESP	0	0	8	5	20	14	0	3	0	10	13	0	6	22	10	0	0	0	3	0	0	0	0	2	0	0
SWE	0	8	12	6	16	3	12	0	9	5	6	0	4	17	0	7	0	1	10	0	0	0	0	0	0	0
UKR	0	2	7	0	4	14	11	4	0	19	0	1	0	22	5	2	6	8	0	0	0	5	0	6	0	0
GBR	7	1	12	14	8	5	2	6	4	0	3	0	13	17	0	12	4	0	5	0	0	0	1	2	0	0

Table 7: Vote Data from the 2018 Eurovision Final

		Candidates																								
		ALB	AUS	AZE	BEL	CYP	CZE	DNK	EST	FRA	DEU	GRC	ISL	ISR	ITA	MLT	NLD	MKD	NOR	RUS	SMR	SRB	SVN	ESP	SWE	CHE	GBR
Voters	ALB	0	3	11	0	7	0	0	0	3	0	6	0	0	13	0	7	18	5	13	10	0	0	0	6	14	0
	AUS	0	0	3	0	0	7	0	0	13	3	0	18	0	5	4	12	7	12	4	0	1	0	0	20	7	0
	AZE	7	2	0	6	3	0	0	0	0	0	6	0	0	11	14	7	11	1	24	10	0	4	2	0	8	0
	BEL	0	2	11	0	8	0	0	0	0	0	3	7	0	10	12	16	10	8	13	1	0	5	0	2	8	0
	CYP	2	2	6	0	0	0	0	0	7	0	24	0	5	16	3	11	0	1	20	0	0	0	4	7	8	0
	CZE	0	8	12	0	0	0	0	2	2	0	0	10	3	10	3	10	7	10	12	0	0	10	0	12	5	0
	DNK	0	2	9	0	0	0	0	10	0	8	0	4	0	4	0	14	10	17	3	0	0	0	1	22	12	0
	EST	0	2	6	1	0	8	8	0	3	0	0	5	0	0	0	15	0	10	18	0	4	7	0	15	14	0
	FRA	0	10	6	0	0	3	5	0	0	0	0	6	12	18	0	17	7	8	3	0	0	0	7	10	4	0
	DEU	0	15	0	0	0	0	5	0	4	0	0	2	0	18	3	15	7	17	8	0	0	3	0	3	16	0
	GRC	8	5	8	0	24	0	0	0	1	0	0	2	0	17	5	6	1	0	18	6	0	4	4	0	7	0
	ISL	0	20	4	0	0	8	4	3	1	0	0	0	0	9	0	12	8	12	0	1	2	0	0	20	12	0
	ISR	0	11	10	0	0	1	1	8	6	0	0	0	0	18	4	14	0	10	12	0	0	0	5	6	10	0
	ITA	13	12	8	0	0	4	16	5	5	0	0	7	0	0	8	5	10	10	8	0	0	0	0	2	3	0
	MLT	2	3	10	0	6	0	0	0	0	0	4	1	0	24	0	17	8	7	14	1	0	0	0	11	8	0
	NLD	0	4	4	0	1	4	12	0	0	0	0	7	0	16	8	0	4	12	5	0	0	0	3	20	16	0
	MKD	20	10	5	0	1	0	0	0	0	0	0	2	0	15	11	14	0	5	6	8	10	2	0	0	7	0
	NOR	0	4	8	0	1	12	9	2	0	0	0	10	0	10	0	15	10	0	1	0	0	0	0	20	12	2
	RUS	3	4	24	15	8	0	0	0	0	0	10	9	0	1	6	5	4	10	0	5	3	6	1	0	2	0
	SMR	7	0	8	0	12	0	0	0	0	0	18	8	1	20	0	9	1	3	22	0	0	0	0	0	7	0
	SRB	0	7	3	0	0	4	2	6	1	0	0	5	0	17	0	3	24	4	8	1	0	10	2	8	11	0
	SVN	0	2	10	0	0	12	7	1	3	0	0	7	0	16	1	11	14	6	0	0	10	0	0	7	9	0
	ESP	0	15	8	0	5	6	0	0	4	0	0	3	0	16	0	16	0	7	4	0	0	1	0	18	13	0
	SWE	0	11	8	0	7	1	7	10	0	0	0	8	0	12	2	18	2	16	3	0	0	0	0	0	11	0
	CHE	10	4	0	0	0	0	1	0	5	6	0	0	0	17	0	16	14	15	3	0	7	0	5	12	0	1
	GBR	0	18	10	0	1	1	9	0	2	0	0	8	0	0	0	10	12	12	4	0	0	0	2	15	12	0

Table 8: Vote Data from the 2019 Eurovision Final

12 References

[1] Jeffrey Ely, Alexander Frankel, and Emir Kamenica. Suspense and surprise. *Journal of Political Economy*, 123(1):215–260, 2015.

[2] Paolo Bizzozero, Raphael Flepp, and Egon Franck. The importance of suspense and surprise in entertainment demand: Evidence from wimbledon. *Journal of Economic Behavior & Organization*, 130:47–63, 2016.

[3] Silvia Knobloch-Westerwick, Prabu David, Matthew S. Eastin, Ron Tamborini, and Dara Greenwood. Sports Spectators' Suspense: Affect and Uncertainty in Sports Entertainment. *Journal of Communication*, 59(4):750–767, 12 2009.

[4] Tim Pawlowski, Georgios Nalbantis, and Dennis Coates. Perceived game uncertainty, suspense and the demand for sport. *Economic Inquiry*, 56(1):173–192, 2018.

[5] Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213, 2005.

[6] Fred Glover and Eric Taillard. A user's guide to tabu search. *Annals of operations research*, 41(1):1–28, 1993.

[7] Manuel Laguna. A guide to implementing tabu search. *Investigación Operativa*, 4(1):5–25, 1994.

[8] Michael Pilegaard Hansen. Tabu search for multiobjective optimization: Mots. In *Proceedings of the 13th international conference on multiple criteria decision making*, pages 574–586. Citeseer, 1997.