Multiple Attribute List Aggregation with Applications in Collaborative Playlist Editing and Job Scheduling

Eyal Briman[1] **and Nimrod Talmon**[1]

[1]Ben Gurion University of the Negev

Abstract. We introduce a multi-objective decision-making model for social choice that integrates time-based constraints, aiming to generate a ranked list that harmonizes both agent preferences (as, e.g., per approval ballots) together with overarching global constraints. First, we analyze the general model, showing that it is generally NP-hard, but admits polynomial-time algorithms for a special case; we also develop heuristic solutions for the general case. Furthermore, we explore potential applications of the model and demonstrate its relevance by focusing on the use cases of democratic playlist editing and democratic job scheduling. In these scenarios, our aim is to generate a list that reflects agent preferences for a given set of musical tracks or a given set of jobs to be processed while also considering soft constraints regarding the sequencing and transitions of tracks or jobs over time. We illustrate how these problems can be translated into our model, and present simulation results where we apply our heuristics to solve specific instances of the problems. We contend that our results are promising – as they are able to strike a balance between the different optimization objectives – not only for these specific use cases, but also for a plethora of other use cases.

1 Introduction

Social choice theory examines collective decision-making processes by aggregating individual preferences to reach a collective choice or outcome [2]. In the standard setting of single-winner elections, a single alternative is chosen from a set of alternatives, based on group preferences. Moving beyond single-winner elections, in multi-winner elections — which formally generalize single-winner elections – a set of candidates or alternatives is selected, following agent preferences [11]. Going even further, in participatory budgeting – which formally generalizes multi-winner elections – a set of projects, each with its cost, is selected while respecting agent preferences and a given budget. We argue that advancements in social choice many times correspond to the desire of having collective decision-making tools that aggregate agent preferences and output *increasingly complex outputs*, taking into account multiple objectives. Our proposed social choice model can be viewed from this angle; and, correspondingly, the aggregation methods that we develop can be seen as computational tools that are able to aggregate agent preferences and output fairly structurally-involved outputs. In particular, we consider a social choice setting that consists of the following ingredients: (1) A set of elements out of which a subset shall be selected (similar to a multi-winner election); (2) where these elements have certain numerically-valued attributes that shall be taken into consideration (formally generalizing participatory budgeting); and (3) where the selected subset of elements shall be ordered (similarly to ranking elements, such as in proportional ranking [28]). We describe our setting formally in Section 2. Essentially, we formulate a multi-objective optimization problem that balances between two considerations: (1) First, the ordered subset (i.e., list) that we output shall respect agent preferences (we model this aspect by aiming to maximize the score of the output list according to some multi-winner voting rule); (2) second, the trend or pattern of the attribute values over time within the output list (we model this aspect by aiming to minimize the distance of the corresponding patterns to "ideal", predefined patterns).

Through the use of mathematical optimization techniques and the development of effective heuristics, our model provides a robust framework for tackling diverse use-cases that involve multi-attribute decision problems in which an output list is to be agreed upon. While in this paper we concentrate on a specific application – namely, of democratic playlist editing – below we first describe several potential applications of our model. Consider the following applications: (1) **Democratic Planning:** Consider a cooperative manufacturing plan for highly logistic complex products, sensitive products, or products with occasional or seasonal demand. We aim to optimize different attributes based on the social choice, ideal demand changing over time, the ideal stock and inventory changing over time, and different measures such as service quality type 1 or 2, which will ideally change over time to minimize costs; (2) **Democratic Media:** Creating content that serves different functions for various attributes over time, for example, a TV series or a movie. The amount of stress/relief or happiness/sadness that the show/movie creates in the viewer's experience as the season or movie evolves can be measured.

In Section 2 we provide our formal model; then, in Sections 6, 7 we describe a different application over which we demonstrate the applicability of our model as well as the suitability and quality of several heuristic solutions that we propose for the model. The concrete applications that we concentrate on in this paper are:

- **Democratic Playlist Editing:** Producing a musical playlist – perhaps to accompany a podcast, movie, TV show, theater or dance, where there should be changes in different attributes over time, such as tempo, loudness, and emotions that will be expressed through the music (using chords, scales, and other musical tools).
- **Democratic Job Scheduling:** the goal is to create a sorted list of jobs using a voting process that aggregates preferences of team members. However, this process must also account for global constraints related to tasks' or features' attributes that change over time. Sequencing these requires prior knowledge that not all team members may possess.

Paper Structure After discussing some related work (in Section 1.1), we go on to describe our formal model (in Section 2). Then, as the aspect of our model that corresponds to respecting agent preferences is both crucial to our model as well as general, in Section 3 we describe how to capture different multi-winner rules in our model. We go on to provide a computational analysis of our general model (in Section 4) and to describe various general heuristics that we propose for solving instances of the general model (in Section 5). We continue to describe the problem of democratic playlist editing and Democratic job scheduling in detail and in a formal way (in Sections 6, 7) and to report on computer-based simulations that we have performed on real-world and artificial data to evaluate the relevance and the quality of our algorithms (in Section 6). We conclude in Section 8 with a discussion on the implications of our research and on promising future research directions.

1.1 Related Work

We mention general related work: (1) As our model can be viewed as a time-based social choice model, we mention ongoing work regarding the aggregation of continuously-changing agent preferences [1]. (2) Second, in our work, we draw inspiration from work that combines social choice aspects with recommendation systems, such as the work of Burke et al. [8], that evolves around the exploration of dynamic fairness-aware recommendation systems using multi-agent social choice. (3) Third, our multi-attribute setting also has some connections with work on the group activity selection problem [9].

Multi-Attribute Social Choice This involves aggregating preferences of a group over alternatives with multiple attributes. Our work focuses on a multi-attribute setting where each element has a vector of numerical attribute values [15, 5, 7]. Various works explore social choice settings with different attributes, such as democratic parliamentary elections aiming for proportional representation of attributes like gender and race in society [27, 4, 6, 22, 23].

Proportional Ranking Traditional approval voting ranks alternatives based on the percentage of approving voters, but it may not accurately reflect the preferences of the population. The proportional ranking model addresses this by interleaving alternatives supported by different groups of agents, reflecting relative popularity. It emphasizes sorting candidates to ensure proportional committees, considering candidate diversity and order. This model finds applications in recommendation systems, hiring, committee elections, and liquid democracy [28]. In our work, we aim to generate rankings with a broader scope than Skowron et al.[28]. Specifically, in Section6, we consider proportionality in collaborative playlist editing, where tracks are ranked based on acoustic features, popularity, and proportional representation. We also mention other related works [18, 14].

Multi-Attribute Scheduling The single-machine scheduling problem involves finding the optimal order of tasks on a single machine to minimize the total completion time or other objectives. Multi-attribute scheduling extends this by considering multiple objectives and constraints. It aims to find a schedule that satisfies constraints while optimizing objectives like completion time or resource utilization [16, 20]. Our model generalizes the multi-attribute [26] multi-agent single-machine problem, emphasizing multi-objective optimization and incorporating the social choice aspect.

2 Multi-Attribute List Aggregation (MALA)

In this section, we present the formal model of Multi-Attribute List Aggregation (MALA). An instance of MALA consists of:

1. A set of y attributes, denoted with their index $q \in [y]$.
2. A set of m elements, $C = \{c_1, \ldots, c_m\}$. Each element c_i, $i \in [m]$, for each attribute $q \in [y]$, has some numerical value; we define c_i^q to be the numerical value of element i for attribute q (so, in particular, $c_i^q \in \mathbb{R}$).
3. A value $k \leq m$, $k \in \mathbb{N}$; this is the desired size of the list that is the output of the instance.
4. A set of z so-called Ω *constraints*, denoted by $\{\Omega_1, \ldots, \Omega_z\}$. Below we describe what is an Ω-constraint: in particular, an Ω-constraint is defined by a tuple (q, F, d, w); next we describe what are q, F, d, and w:
   - $q \in [y]$ is the index of some attribute.
   - $F := \{f_1, \cdots, f_t\}$ is a family of t vectors, each of length k; formally, $f_\ell \in \mathbb{R}^k$, $\ell \in [t]$. We use a square brackets notation for vectors; i.e., for $\ell \in [t]$, $s \in [k]$, we write $f_\ell[s]$, $f_\ell[s] \in \mathbb{R}$, to denote the value of the s'th element of f_ℓ. (Intuitively, each of these vectors corresponds to some ideal behavior of the output list with respect to attribute q).
   - d is a metric returning a distance between two real-valued vectors of length k. Formally, $d : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$; i.e., d is a function that takes two vectors of length k and returns a numeric value that we interpret as their distance; and it shall be a metric. (Intuitively, the metric d quantifies how close-to-ideal is the output list to at least one of the ideal vectors, with respect to the attribute q; note that we will use d only to evaluate the distance between some possible solution and some vector of F.)
   - $w \in \mathbb{R}$, is the weight that the Ω-constraint gets. (Intuitively, it corresponds to the importance of that Ω-constraint.)

An instance of MALA as described above defines a cost for each possible $solution$ to it. To describe what it is, let $solution$ be some possible solution to an instance of MALA; first, formally, $solution$ shall satisfy the following:

- $solution \in C^k$; i.e.,., $solution$ is a vector of k elements, each from C.
- For $s_1 \neq s_2$ (two different indices in $solution$) , it holds that $solution[s_1] \neq solution[s_2]$; i.e.,., there can be no repetitions.

Below we describe the *cost* of a possible solution $solution$:

- First, the cost of a solution $solution$ with respect to a specific Ω-constraint $\Omega = (q, F, d, w)$ is, roughly speaking, the weight (w) multiplied by the distance (according to d) between the values of the elements of the solution for the attribute q to the vector of F that is the closest to it; formally, we define:

$$cost(solution, \Omega) = w \cdot min_{f \in F} d(f, solution^q) \ ,$$

  where $solution^q \in \mathbb{R}^k$ is the vector containing the values of the elements of the solution with respect to the attribute q; formally, $solution^q[s] := solution[s]^q$, $s \in [k]$.
- Second, the cost of a solution $solution$ with respect to an instance of MALA $MALA$ (that contains z Ω-constraints) is defined naturally as the summation of its cost with respect to each of the Ω-constraints; formally, we define:

$$cost(solution, MALA) = \sum_{i \in [z]} cost(solution, \Omega_i) \ .$$

Given an instance of MALA, denoted by $MALA$, we are looking for a solution $solution$ of minimum cost; formally, we are looking for the following:

$$\arg \min_{\substack{solution \in C^k \\ solution[s_1] \neq solution[s_2] \\ \text{for } s_1 \neq s_2}} cost(solution, MALA)$$

[1]

Example 1. *Consider the following toy example: Jimmy would like to prepare food to take to work and needs to decide what to bring to each of his three meals, during the day. His decision is based on 3 attributes: (1) personal preferences, (2) calories, (3) and sugar. His candidates are: (1) apple, (2) orange , (3) omelette sandwich , and (4) tuna sandwich. Each candidate is defined by its unique attributes' values. Jimmy also sets his ideal attribute values for each one of the three meals and the importance of each attribute. Thus, the formal instance – denoted by $MALA$ – is given by (with some specific data): $y = 3$; $k = 3$; and the following constraints: q_1= apple, q_2= orange, q_3= omelette sandwich, q_4= tuna sandwich. $F_1 = \{f_1\}$, where $f_1[1] = 10, f_1[2] = 10, f_1[3] = 10$; $F_2 = \{f_2, f_3\}$, where $f_2[1] = 80, f_2[2] = 600, f_2[3] = 60$, and where $f_3[1] = 100, f_3[2] = 500, f_3[3] = 80$; $F_3 = \{f_4, f_5\}$, where $f_4[1] = 0, f_4[2] = 20, f_4[3] = 0$, and where $f_5[1] = 0, f_5[2] = 0, f_5[3] = 0$; d_i is the ℓ_1 norm, for each $i \in [3]$; $w_1 = 1, w_2 = 0.5, w_3 = 0.8$. $C = \{c_1, c_2, c_3, c_4\}$ with jimmy's preferences given by: c_1^1 = 7, c_1^2 = 95, c_1^3 = 19, c_2^1 = 4, c_2^2 = 60, c_2^3 = 12, c_3^1 = 5, c_3^2 = 530, c_3^3 = 15, c_4^1 = 10, c_4^2 = 570, c_4^3 = 50;*

A possible $solution_1$ may be: [apple, omelette sandwich, orange] with cost: $cost(solution_1, MALA) = \sum_{i=1}^{3} w_i \cdot (min_{f_\ell \in F_i} d_i(f, solution^{q_i}) = 1 \cdot (|10-7| + |10-5| + |10-4|) + 0.5 \cdot (|100-95| + |500-530| + |80-60|) + 0.8 \cdot (|0-19| + |20-15| + |0-12|) = 70.3$.

3 Committee Scoring Rules Using MALA

Next, we will demonstrate the versatility of the MALA model for various voting rules. Our model is specifically designed for democratic settings, making it applicable to a wide range of use cases, including committee elections. This will be useful for both the application of the democratic playlist editing and the application of the democratic job scheduling, which we will discuss in detail later.

Approval Voting and Borda Count: In the case of selecting a committee, the ideal approval score or Borda count would be achieved if all n agents voted or ranked the same m candidates, indicating a consensus. To model this, we represent the perfect approval score and Borda count as a vector of size k where each candidate's score is a constant value representing the number of votes or ranking positions they received from the n agents. Thus, this vector serves as an upper bound for calculating the cost of a given solution. Formally, we define: (1) Each candidate's voting score is given by: voting score$(c_1) = \sum_{i=1}^{n} c_{i,1} \cdots$, voting score$(c_m) = \sum_{i=1}^{n} c_{i,m}$, and $c_{i,j} \in 0, 1$ or $c_{i,j} \in 0, 1, \cdots, k$ indicates the approval or ranking of candidate $j \in [m]$ by agent $i \in [n]$. (2) $F_1 = f_1$, where in approval voting $f_1[s] = n$, and in Borda count $f_1[s] = n \cdot k$, for all $s \in [k]$. (3) d = any metric distance, such as ℓ_1 or ℓ_2.

[1] It is worth noting that we assume all attributes can be treated as numerical, as categorical attributes can often be transformed into numbers. However, it would also be interesting to experiment with labeled data that cannot be easily transformed into numerical values.

PAV Score: The PAV (Proportional Approval Voting) score assigns scores to candidates based on the number of votes they receive, with the goal of allocating seats to candidates proportionally to their support, while also considering the number of available seats. In our model, each agent approves or disapproves of certain elements. We introduce a PAV cost constraint to represent the "loss" of the potential PAV score. This model can also be extended to OWA-based rules [13]. For each agent $v_j \in v_1, \cdots, v_n$ voting on elements $c_1, \cdots, c_m$, we create an Ω-constraint, and its distance will be the "PAV-cost," reflecting the "loss" of the potential PAV score. The following definitions apply: (1) $q \in [m]$ corresponds to every agent. (2) $solution[s]^j$ = 1 if agent j approves of candidate i and 0 otherwise, for all $solution \in C^k$. (3) $F = \{f_1\}$, $f_1 = \{[1]^k\}$. (4) $d(x, y) = \sum_{j=1}^{k-\ell_1(x,y)} \frac{1}{j}$, $\ell_1(x, y) < k$ and 0 otherwise. Given such individual agent Ω-constraints, adding a weight vector of all "1" results in the realization of PAV as an instance of MALA.

4 Computational Analysis

To study the computational complexity of MALA we consider its decision variant, in which we are given an additional input that is the maximum total cost for which existence of a solution above is to be decided. First, we observe that, following the formulation described above of PAV as a MALA instance, NP-hardness is established [3]. Next we show that MALA is also NP-hard even with only 2 constraints.[2]

Theorem 1. *The MALA Decision Problem is NP-hard.*

Proof. We provide a reduction from the subset-sum problem [21], where an instance X containing x_i, $i \in [n]$ is a "yes-instance" if a subgroup $X' \subset X$ exists that satisfies $|X'| = \frac{n}{2}$ and $\sum_{x_i \in X'} x_i = \frac{\sum_{x_i \in X} x_i}{2} = \frac{B}{2}$. To build an input for the MALA decision problem given the subset-sum input, we set the following Ω-constraints:

- **q** - We have a MALA problem with two identical attributes: q_1, q_2 having $c_i^1 = c_i^2 = x_i$ for all i.
- **F** - We set $F_1 = f_1$ where $f_1^j = 0$, and $F_2 = f_2$ where $f_2^j = M$, for all $j \in [k]$ (vector's length), where $M = \sum_{x_i \in X} x_i$.
- **d** - We create two distances based on the ℓ_1 distance between two given vectors:

$$d_1(vector_1, vector_2) = \begin{cases} 1, & \ell_1(vector_1, vector_2) > \frac{B}{2} \\ 0, & \ell_1(vector_1, vector_2) \leq \frac{B}{2} \end{cases}$$

$$d_2(vector_1, vector_2) = \begin{cases} 1, & \ell_1(vector_1, vector_2) > \frac{M \cdot n}{2} - \frac{B}{2} \\ 0, & \ell_1(vector_1, vector_2) \leq \frac{M \cdot n}{2} - \frac{B}{2} \end{cases}$$

- **Weights**- We set $w_1 = w_2 = \frac{1}{2}$.

We formulate the MALA model as a decision problem - Given all candidates C and: Ω-constraints, we want to determine if there exists a subset $X' \subseteq X$ such that $\sum_{i=1}^{2} cost(X', \Omega_i) = 1$. If such a subset X' exists, then $\sum_{x_i \in X'} x_i = \frac{\sum_{x_i \in X} x_i}{2} = \frac{B}{2}$. Conversely, if $c_{i,1} = c_{i,2}$, for all i, $F_1 = F_2$, $d_1 = d_2$, and $X = q_1 = q_2$, we can

[2] There is a delicate point here with respect to the representation of the input. We discuss consequences of this to different applications in Section 8, but here, for the formal hardness statement and proof, it is crucial to describe the representation of the input that affects the length of the input. So, in particular, it is sufficient to assume that the F vectors in the input are given explicitly, while the d metrics are given as black-boxes of length $O(1)$.

reduce the problem to the subset sum problem where $|X| = n$. In this case, if there exists a subset $X' \subseteq X$ such that $|X'| = \frac{n}{2}$ and $\sum_{x_i \in X'} x_i = \frac{\sum_{x_i \in X} x_i}{2} = \frac{B}{2}$, then $\sum_{i=1}^{2} cost(X', \Omega_i) = 1$. Thus, the MALA decision problem is NP-hard even with just two attributes, contradicting our polynomial assumption of the problem. □

5 Algorithms

Since our problem has been shown to be NP-hard, we have developed several heuristic algorithms to obtain a good solution within a reasonable time frame. In this study, we have chosen to test two main algorithms: Genetic and Simulated Annealing. Both of these heuristics are suitable for solving similar combinatorial optimization problems that have complex search spaces and multiple objectives. To simplify the testing process, we make the assumption that each vector family, denoted as F, which describes optimal behavior over time or sequence of some feature q, has a finite set. In each of the heuristics explained here, we aim to find the ordered sub-group of k elements out of a total group of m elements that would minimize the weighted summation of costs defined by Ω-constraints.

Genetic Algorithm - The algorithm begins by generating an initial population of candidate solutions, each represented as a set of parameters. The cost of each solution is calculated based on the problem at hand. The population is then sorted based on the descending cost, and the best solution is identified. In each iteration, the algorithm updates the population size using adaptive population sizing techniques, which adjust the number of solutions in the population based on their performance. The crossover and mutation probabilities, which control the exploration and exploitation of the search space, are also updated adaptively [24]. New solutions are generated through crossover or mutation operations, ensuring that there are no repetitions among the solutions. The cost of each new solution is calculated, and the population is sorted again. If the best solution in the new population has a lower cost than the current best solution, it is updated accordingly. The algorithm continues iterating until the specified total run time is reached. Finally, the best solution found throughout the iterations is returned as the output of the algorithm.

Simulated Annealing - The algorithm starts by generating an initial random solution. It then sets the initial cooling rate and temperature, which are problem-dependent and determine the exploration-exploitation balance. Additionally, a number of iterations for a random start are specified to allow for more diverse exploration. During each iteration, the algorithm calculates the cost of the current solution and compares it to the minimal cost found so far. If the current cost is lower, the minimal cost is updated accordingly. The algorithm also computes the probability of accepting a worse solution based on an adaptive cooling rate [19]. If the probability allows accepting a worse solution, the minimal cost is updated. To explore the search space, a random element and index are generated. If the generated element is included in the solution, it is swapped with the element at the generated index. Otherwise, the element at the generated index is replaced with the generated element. The temperature is decreased using the cooling rate, gradually reducing the exploration ability of the algorithm. The process continues until the specified total run time is reached. Finally, the best solution found throughout the iterations is returned as the output of the algorithm. [3]

[3] The full implementation can be found here https://github.com/EyalBriman/MALA

6 The Democratic Playlist Editing Problem

The focus now shifts to the democratic playlist editing issue, specifically the problem of creating a playlist with a specific logic or theme. This problem, known as the Democratic Automated Generation Playlist Problem, involves a group of friends attempting to create a playlist. In this section, we will discuss this problem and its formulation using MALA. The Automated Generation Playlist Problem involves creating a playlist by selecting tracks from a given list based on their musical attributes, such as scale, key, tempo (beats per minute), time signature, loudness, valence (optimism), danceability, and more. The Democratic Automated Generation Playlist extends the original problem by allowing a community to vote on whether to include tracks in a playlist. Playlist editors and critics usually look for the following three measures in a playlist [17]:

1. **Coherence of tracks** - Listeners tend to like playlists with tracks that correspond (musically and lyrically) to each other homogeneously [25]. In order to model the coherence of a feature using MALA, we must find the vector of some constant value. It will serve as a reference to measure the extent of the coherence of the attribute's behavior over time or over a sequence of events. Thus, we need to search all the positive constant vectors in order to find the most suitable one for $Solution[q]$ over time. Formally: (1) q corresponds to c_i^q. (2) $F = \{f_1, f_2, \cdots, f_t\}$ having $f_i[s] = p, i \in [t], p \in \mathbb{R}$, for all $s \in [k]$. (3) $d = \ell_1$ or ℓ_2.
2. **Smooth transitions between two consecutive tracks** - Smooth transitions are highly valued by users as they provide a seamless progression of attributes, whether in sequence or over time. The primary objective of these transitions is to maintain a consistent flow while minimizing abrupt changes in attributes value's direction over time: (1) $q_{j \in 0,\cdots,k-1}$ address a particular attribute, and e, the maximum explicit number of direction changes it values can undergo. (2) $F = \{f_1, f_2, \cdots, f_t\}$ having $f_i = \mathbb{R}^k, i \in [t]$, such that $0 \leq \sum_{s=2}^{k-1} \delta(sign(f_i[s] - f_i[s-1]), sign(f_i[s+1] - f_i[s])) \leq e \in [k-1]$, $\delta(x, y) = 0$ if $x = y$, else $\delta(x, y) = 1$. (3) $d = \ell_1$ or ℓ_2. This modeling of smooth transitions as well as coherence, is in contradiction to our initial assuming of a finite set of functions F. Because the modeling of these qualities depend on the attribute's behaviour and the actual suggested tracks; we do not have a way of predicting what would be the ideal set of functions, and so we can only approximate by giving a few reasonable functions (a finite set of vectors).
3. **Diversity** -Like many democratic parliaments that ensure seats for different groups in society (such as women and minorities) to maintain a proportional representation of society, a playlist should aim for representation of different attributes of the given tracks, including genres (e.g., jazz, pop, rock, reggae, Brazilian, Afro-beat, Indian), scales (e.g., major and minor), time signatures (e.g., even and odd beat division of a track), and ranges of tempo (e.g., Largo (very slow), Adagio (slow), Andante (medium-slow), Moderato (medium), Allegro (medium-fast), and Presto (fast)).While the two measures before where based on numerical valued attributes, this measure is based on categorical valued attributes. Formally: (1) q corresponds to $c_i^q \in [r]$ having $r = |P^q|$, and P^q to be the set of categories associated with attribute q. (2) $F = \{f_0, f_1, \cdots, f_t\}$, $f_i = \{f_i[s] | s \in k, f_i[s] \in [r], freq(f_i[s], f_i) = \pi_{p \in P^q}\}$ having π_p to be the optimal proportion of each category of attribute q. (3) $d = \ell_1$ or ℓ_2

Experimental Design and Analysis To evaluate the quality of the heuristics discussed earlier, we conducted a simulation of the Demo-

cratic Playlist Editing problem modeled as a MALA-optimization problem. Next, we generated ten 700-track playlists from Spotify's "Top 10,000 Songs Of All Times" playlist[4]. For each instance, we applied the discussed algorithms to find an ordered sub-list of 250 tracks that minimized the cost within a ten-minute run. The heuristic parameters were set as follows: simulated annealing and sequential simulated annealing with a temperature of 1000 degrees, an initial cooling rate of 0.003, and a random start every 1000 iterations. Genetic Algorithm was initialized with an initial crossover probability of 0.85, an initial population size of 100, and a maximum population of 5000. The costs were normalized using a 100-random algorithm, which generated 100 random permutations and selected the one with the minimal cost. This allowed us to calculate the average cost per minute for all 10 instances. We selected three audio features from Spotify's API out of 13 available features [10]: Energy (0.0 to 1.0 score representing intensity and activity), Tempo (measured in beats per minute), and Danceability (0.0 to 1.0 score representing suitability for dancing). These features were chosen for their significant impact on playlist formation. Approval scores were added to each track using an artificial society of 20 agents. An algorithm was used to generate a list of 700 integers, ensuring a sum of 5000. The algorithm randomly and uniformly generated approval ballots, divided the remaining sum by the remaining iterations, and updated the list accordingly. If there was a remaining sum, it was distributed incrementally to randomly selected indices until reaching zero. Next, we generated 10 families of functions (represented as vectors), each one containing 50 different 250-sized vectors. These functions within each family can be divided into two types:

1. We aimed for an ideal behavior of certain attributes over time, specifically smooth transitions for BPM and energy, with 1-3 direction changes to ensure smoothness along the playlist. This resulted in a total of 6 families of functions.
2. We defined an ideal static value to measure the coherence of attributes, where changes in direction are 0 and the slope (i.e., the size of change between two consecutive tracks) is 0. This applies to BPM, danceability, energy, and approval score, with the latter only including one vector/function of "all 20s".

We developed an algorithm that takes in the minimum number of direction changes, a range for the first variable in the vector (distributed uniformly), a range for the slope between consecutive variables in the vector (distributed uniformly), and the minimum and maximum values to set the range of legal values in the vector. The algorithm generates the initial direction (+ or -) uniformly and k indexes in the range of 2-248, where k is the number of direction changes and $index_i - index_{i+1} \geq 2$. Whenever the algorithm reaches one of the indexes, or if the value of the current variable is greater than the maximum value or smaller than the minimal value, a change of direction will occur. Finally, we set the distance d as ℓ_1 and generated a weight for each Ω-constraint combination, including:

1. Energy weights for 0, 1, 2, and 3 changes of direction over time. These weights were generated uniformly between 1 to 3, taking into account the involvement of energy in creating a playlist.
2. Tempo weights for 0, 1, 2, and 3 changes of direction over time. These weights were generated uniformly between 0.0001 to 0.001, as tempo was deemed to be a feature of less importance in our simulation.
3. Danceability weight for 0 changes of direction over time. These weights were generated uniformly between 3 to 4, aiming to create a highly danceable playlist.
4. Approval score weight for 0 changes of direction over time. These weights were generated uniformly between 4 to 5, with the intention of creating a playlist of popular tracks.

We also tested a greedy algorithm as an additional reference, wherein we preformed a local search for the most suitable track to fit into every location in the list.

We executed an additional simulation to determine the time needed for the simulated annealing algorithm to reach half the value obtained by the 100-Random algorithm across 10 unique instances, each consisting of 700 tracks. The variation included selecting and ordering musical tracks ranging from 30 to 240 at intervals of 30.

Results and Analysis The results show that the simulated annealing algorithm achieved the lowest normalized cost quickly, with genetic algorithms performing worse; and confirms that the simulated annealing algorithm takes longer to reach half the value of zRandom as the instance size increases. These results suggest that the simulated annealing algorithm explores a wider range of solutions compared to the more restrictive genetic algorithm, which converges slower due to maintaining parental order. Alternatively, the suboptimal tuning of initial parameters, such as adaptive crossover probability and population size, may explain the genetic algorithm's performance and could be improved with additional tuning techniques. These results suit the findings of Piotr Faliszewski et al [12] on effective heruistics for committee scoring rules.

7 Democratic Job Scheduling

Next we explore another application amenable to modeling using MALA: the Democratic Job Scheduling problem. While the setting may be more general, we concentrate here on a specific scenario for concreteness. Our scenario involves the collaborative scheduling of jobs among n agents, each equipped with their own preferences regarding the prioritization of the list of k unit-time jobs (i.e., all jobs share a uniform execution time t). from a total set of m jobs; in particular, we consider the case where their preferences are expressed as a ranked ballot – i.e., each agent can articulate what is her most-preferred job, second-preferred job, etc. The job scheduling process must factor-in the agents' preferences while adhering to overarching constraints that dictate the preferred resource behavior associated with the progression of each job in the scheduling. In this particular setting, we address a scheduling task within a single assembly line equipped with M machines. Each job requires a dedicated machine, allowing only one job to be serviced at any given time across the M machines. Each of the m job has the following attributes: (1) cost of executing the job (say, in dollars); (2) power consumption (electricity; say, in kilowatts), of running the job on the dedicated machine; (3) labor requirement to execute the job, that is, the number of workers needed to execute the job (say that we only have one type of workers); (4) the machine ID needed for the job execution.

Besides these internal properties of each job, as we have ranked ballots (i.e., ordinal ballots) of all voters to each job, we also have an additional property for each job, namely, the Borda count for the job, calculated using the ranked ballots. In this method (as described above), the higher an agent values a job, the higher the job will be ranked, resulting in a higher score within the range of 1 to k.

We have some constraints and required scheduling properties and values: **Cost:** Regarding the cost, our aim is to minimize the over-

[4] https://open.spotify.com/playlist/1G8IpkZKobrIlXcVPoSIuf

all cost associated with the job scheduling. **Power Consumption:** Regarding the power consumption, we seek some coherence in the power consumption, aiming for consistency within the specified range (this is similar in spirit to what is discussed in the section 6). **Labor Requirements:** We wish the scheduling of jobs to correspond and harmonize with the available labor force allocated. Essentially, we assume that, in each timestep, we have some available work force, and our aim is to closely match the number of workers for each time step with the predetermined working arrangement (i.e., the specified number of available workers at every stage during the shift). If a job requires fewer workers than available in the assembly line, then it results in a waste of manpower; while, conversely, if a job requires more workers than available, then it necessitates additional workforce recruitment, which inevitably escalating costs. **Maximal Job Requirements for Each Machine:** We assume that each machine has a predefined limit of μ jobs that can be executed before a maintenance break must occur. And that, after reaching μ jobs on that machine, there must be a break of τ jobs to allow effective maintenance, otherwise the maintenance will occur parallel to the execution of jobs, resulting in some fine, which we wish to minimize. **Most Ranked Jobs:** Regarding voter preferences, we aim to position jobs based on their Borda count, where higher-ranked jobs take precedence and are executed earlier in the schedule.

MALA Formulation The requirements described above can be translated naturally to MALA constraints: in the modeling formulated below, essentially, any deviation from these specified ideal behaviors would result in fines, weighted by certain weights, W. Concretely, global constraints, representing the ideal behavior of these attributes over time, are enforced through MALA given a solution, denoted as a k-sized vector of jobs, $solution = [job_1, job_2, \ldots, job_k]$. Following this intuition, below we show a formulation of the Democratic Job Scheduling usecase to the framework of MALA: **Ideal Cost:** Let q denote a cost. $job^q \in \mathbb{R}$ represents the cost associated with each job; $F = \{f_1\}$ with $f_1 = 0^k$ denotes the constraint ensuring minimal job costs; $d = \ell_1$ or ℓ_2. **Ideal Power Consumption:** Let q denote kilowatts. $job^q \in \mathbb{R}$ represents the power consumption (in kilowatts) requirement for each job; $F = \{f_1, \ldots, f_z\}$ where $f_{i\in[z]} = b^k$. $b \in \{0, \ldots, h\}$ denotes a constant value between 0 kW and the upper bound of recommended power consumption, ensuring a consistent power consumption pattern; $d = \ell_1$ or ℓ_2. Defines the distance metric concerning power consumption. **Labor Constraints:** Let q denote the total number of workers. $job^q \in \mathbb{N}$ represents the required number of workers for each job. $F = f_1$ where $f_1 = \mathbb{N}^k$, indicating the count of available workers for each job in a list of k jobs. Here, f_1 outlines the shift workforce plan, defining a shift as $k \times t$, where t represents the uniform execution time for each job. $d = \ell_1$ or ℓ_2. This parameter defines the distance metric concerning labor constraints. **Maximal Job Requirements for Each Machine:** For each machine i among $M \in \mathbb{N}$ machines: Let q_i denote the activation for machine i. Here, $job^{q_i} \in \{0, 1\}$ denotes whether machine i is needed by job j. This implies for every job there are $M - 1$ machines, resulting in attributes where $job^q = 0$ and one case where $job^q = 1$, as for every job there is exactly one machine needed. $F_i = \{f_1, \cdots, f_t\}$ where $f_{i\in[t]}^{j\in[k]} \in \{0, 1\}$ encompasses all possible permutations ensuring a limit of μ_i jobs followed by a setup time of τ_i jobs. So, after μ_i occurrences of 1s (not necessarily consecutive) , there appears τ_i occurrences of consecutive 0s. $d_i = \ell_1$ or ℓ_2. This defines the distance metric concerning machine constraints for each machine i.

Experimental Analysis (Scheduling) In a simulated scenario involving a single assembly line equipped with 5 distinct machines, a total of 750 jobs were randomly selected from a pool of 5000 artificial jobs. Each job is uniquely identified and defined by a set of attributes as follows: Cost: Randomly distributed between 100 and 1000 uniformly; Power Consumption: Randomly distributed between 10 and 100 uniformly; Worker Requirement: Randomly distributed between 1 and 10 uniformly; Machine Assignment: Each job is assigned to one of the five machines, determined randomly between machine IDs 1 and 5.

Ideal vectors, denoted as F, encapsulating all these attributes were randomly generated within the ranges expressing the values distributed the same way as outlined for the job attributes. For each machine, 1000 potential permutations were randomly sampled representing the maximal jobs that can be executed on the machine before it needs a setup. These permutations ensure a specified sequence where the maximum number of jobs that can be executed (represented by μ, i.e., non-consecutive 1s) is followed by the number of jobs required to reset the machine (denoted by τ, i.e., consecutive 0s). μ is uniformly generated between 30 and 60, while τ is uniformly generated between 10 and 25. Additionally, a group of 20 artificial where generated represented by generated randomly rankings of 250 jobs out of the 750 jobs in such way that ensured that $\sum_{job=1}^{750} c(i, \text{job}) = \sum_{j=1}^{250} j$ for each agent $i \in [n]$. These rankings were employed to calculate the Borda count for each job. Given the computational complexity of this problem and the necessity to find an optimal solution within a reasonable time frame, heuristic methodologies were applied. Specifically the same four heuristic algorithms were utilized, as those employed in the democratic playlist use-case: simulated annealing, genetic algorithms, a greedy algorithm and a z-random algorithm (having $z = 100$). We created 10 instances (i.e., 750 jobs randomly chosen from the 5000 jobs) where the task is to find a 250 sized solution. We executed the same procedure as explained in sub section 6.

Finally, we executed an additional simulation to determine the time needed for the simulated annealing algorithm to reach half the value obtained by the 100-Random algorithm across 10 unique instances, each consisting of 700 jobs. The variation included selecting and ordering jobs ranging from 25 to 150 at intervals of 25.

Results and Analysis The findings indicate that the simulated annealing algorithm rapidly achieved the lowest normalized cost, whereas genetic algorithms performed less favorably. These results reinforce the earlier conclusions drawn from the democratic playlist use-case; and, as the instance size increases, the time taken for simulated annealing to reach half of the 100-Random score also increases. This observation aligns with the previous findings in the Democratic Playlist Editing use-case. However, a notable distinction arises: not only does it take more time to reach solutions equal to or smaller than half the cost of 100-Random, but this time duration significantly escalates with the instance's size. This suggests that the solution space in the Democratic Scheduling use-case is notably more intricate.

8 Discussion and Outlook

We introduced the MALA framework of multiobjective optimization, designed to establish an ordered set of candidates while striving to discover a solution that minimizes the weighted summation of distances from the ideal behaviors of various attributes as they evolve over time and demonstrated its applicability.

In future research, we suggest exploring: adding logical constraints; a time-axis; several dimensions; other heuristic solutions;

a careful computational analysis of special cases of MALA; further experiments; and further, diverse application domains.

References

[1] Anonymous. Continuous preference aggregation in one dimension, 2023.

[2] K. J. Arrow, A. Sen, and K. Suzumura. *Handbook of Social Choice and Welfare*, volume 2. Elsevier, Cambridge University Press, 2010.

[3] H. Aziz. Proportional representation in approval-based committee voting and beyond, 2018. URL https://arxiv.org/abs/1802.00882.

[4] H. Aziz. A rule for committee selection with soft diversity constraints, 2018. URL https://arxiv.org/abs/1803.11437.

[5] W. Bossert and H. Peters. Multi-attribute decision-making in individual and social choice. *Mathematical Social Sciences*, 40(3):327–339, 2000. URL https://EconPapers.repec.org/RePEc:eee:matsoc:v:40:y:2000:i:3:p:327-339.

[6] R. Bredereck, P. Faliszewski, A. Igarashi, M. Lackner, and P. Skowron. Multiwinner elections with diversity constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] R. Burke, N. Mattei, V. Grozin, A. Voida, and N. Sonboli. Multi-agent social choice for dynamic fairness-aware recommendation. In *Adjunct Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization*, UMAP '22 Adjunct, page 234–244, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392327. doi: 10.1145/3511047.3538032. URL https://doi.org/10.1145/3511047.3538032.

[8] R. Burke, N. Mattei, V. Grozin, A. Voida, and N. Sonboli. Multi-agent social choice for dynamic fairness-aware recommendation. In *Adjunct Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization*, pages 234–244, 2022.

[9] A. Darmann, E. Elkind, S. Kurz, J. Lang, J. Schauer, and G. Woeginger. Group activity selection problem. In *Internet and Network Economics: 8th International Workshop, WINE 2012, Liverpool, UK, December 10-12, 2012. Proceedings 8*, pages 156–169. Springer, 2012.

[10] D. Duman, P. Neto, A. Mavrolampados, P. Toiviainen, and G. Luck. Music we move to: Spotify audio features and reasons for listening. *PLOS ONE*, 17(9):1–18, 09 2022. doi: 10.1371/journal.pone.0275228. URL https://doi.org/10.1371/journal.pone.0275228.

[11] P. Faliszewski, P. Skowron, A. Slinko, and N. Talmon. Multiwinner voting: A new challenge for social choice theory. *Trends in computational social choice*, 74(2017):27–47, 2017.

[12] P. Faliszewski, M. Lackner, D. Peters, and N. Talmon. Effective heuristics for committee scoring rules. In *Proceedings of the AAAI Conference on ArtificialIntelligence*, volume 32, 2018.

[13] P. Faliszewski, P. Skowron, A. Slinko, and N. Talmon. Committee scoring rules: Axiomatic characterization and hierarchy. *CoRR*, abs/1802.06483, 2018. URL http://arxiv.org/abs/1802.06483.

[14] J. Goldsmith, J. Lang, N. Mattei, and P. Perny. Voting with rank dependent scoring rules, 2014.

[15] S. Gupta, P. Jain, and S. Saurabh. Well-structured committees. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 189–195, IJCAI Yokohama Japan, 7 2020. International Joint Conferences on Artificial Intelligence Organization. doi: 10.24963/ijcai.2020/27. URL https://doi.org/10.24963/ijcai.2020/27. Main track.

[16] S. K. Gupta and J. Kyparisis. Single machine scheduling research. *Omega*, 15(3):207–227, 1987. ISSN 0305-0483. doi: https://doi.org/10.1016/0305-0483(87)90071-5. URL https://www.sciencedirect.com/science/article/pii/0305048387900715.

[17] S. Ikeda, K. Oku, and K. Kawagoe. Analysis of music transition in acoustic feature space for music recommendation. In *Proceedings of the 9th International Conference on Machine Learning and Computing*, ICMLC 2017, page 77–80, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348171. doi: 10.1145/3055635.3056602. URL https://doi.org/10.1145/3055635.3056602.

[18] J. Israel and M. Brill. Dynamic proportional rankings. *arXiv preprint arXiv:2105.08043*, 2021.

[19] M. Karabin and S. J. Stuart. Simulated annealing with adaptive cooling rates. *The Journal of Chemical Physics*, 153(11):114103, 2020. doi: 10.1063/5.0018725. URL https://doi.org/10.1063/5.0018725.

[20] D. Karger, C. Stein, and J. Wein. Scheduling algorithms, 2010.

[21] R. M. Karp. *Reducibility among combinatorial problems*. Springer, Berkeley University, California, USA, 1972.

[22] J. Lang and P. Skowron. Multi-attribute proportional representation. *Artificial Intelligence*, 263:74–106, 2018. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2018.07.005. URL https://www.sciencedirect.com/science/article/pii/S0004370218304089.

[23] J. W. Lian, N. Mattei, R. Noble, and T. Walsh. The conference paper assignment problem: Using order weighted averages to assign indivisible goods, 2018.

[24] F. G. Lobo and C. F. Lima. A review of adaptive population sizing schemes in genetic algorithms. In *Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation*, GECCO '05, page 228–234, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 9781450378000. doi: 10.1145/1102256.1102310. URL https://doi.org/10.1145/1102256.1102310.

[25] S. Pauws and B. Eggen. Realization and user evaluation of an automatic playlist generator. *Journal of New Music Research*, 32(2):179–192, 2003. doi: 10.1076/jnmr.32.2.179.16739. URL https://www.tandfonline.com/doi/abs/10.1076/jnmr.32.2.179.16739.

[26] P. Perez-Gonzalez and J. M. Framinan. A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, 235(1):1–16, 2014.

[27] S. K. Sikdar. *Optimal multi-attribute decision making in social choice problems*. Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180, USA, 2018.

[28] P. Skowron, M. Lackner, M. Brill, D. Peters, and E. Elkind. Proportional rankings, 2016. URL https://arxiv.org/abs/1612.01434.