List Aggregation of Ranked Preferences

Nihar B. Shah
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
nihar@eecs.berkeley.edu

DRAFT: Oct. 29, 2013

Abstract

We consider the problem of aggregating a given set of orderings (rankings) into a (smaller) set of aggregate orderings that maintains certain properties such as fairness. Classical results such as Arrow’s impossibility theorem show the impossibility of doing so with just one aggregate ordering. This work thus considers doing so with a ‘list’ of aggregate orderings, and in particular, with a list of the smallest possible size, under a (stronger) notion of fairness. Upper and lower bounds on the size of such a list are derived, and these bounds are a logarithmic factor away from each other. The algorithm presented for aggregating these orderings also forms the basis of a new game ‘Match to Race’ available online.

I. INTRODUCTION

Consider \( n \) candidates and a set of rankings, with each element of this set being a ordering of some of these candidates. Consider the problem of aggregating this (possibly large) set of orderings into a representative ordering of the candidates. Classical results such as Arrow’s impossibility theorem \(^1\) prove the impossibility of producing a single aggregate ordering that satisfies certain ‘fairness’ criteria. Several works such as \(^2\)–\(^4\) thus consider the problem of deriving an aggregate ordering that forms the ‘best fit’ (under different metrics) for the input rankings. This writeup, however, pursues a different direction: we consider aggregating the given rankings into a list of orderings that is consistent with the given rankings, and in particular, aim to find the minimum size of such a list.

We now describe the problem more formally. Consider \( n \) candidates \( \{a_1, \ldots, a_n\} \). Consider an arbitrary set of orderings \( V_I \) given as an input. Each element of set \( V_I \) is a total ordering of some arbitrary subset of the \( n \) candidates. We say that candidate \( a_i \) beats candidate \( a_j \) in the set \( V_I \) if the number of orderings in \( V_I \) that rank \( a_i \) strictly higher than \( a_j \) is strictly larger than the number of orderings in \( V_I \) that rank \( a_j \) strictly higher than \( a_i \).

The goal is to aggregate the orderings in \( V_I \) into an output set of orderings \( V_O \) such that:

- **Fairness**: for any two candidates \( a_i \) and \( a_j \), if \( a_i \) beats \( a_j \) in \( V_I \) then \( a_i \) must also beat \( a_j \) in \( V_O \).
- **Completeness**: each element of the output set \( V_O \) is a complete ordering of the \( n \) candidates. Furthermore, the ordering is strict (i.e., has ‘\( > \)’ and not ‘\( \geq \)’).
- **Efficiency**: the size of the set \( V_O \) should be small.

The goal is to construct the most efficient (i.e., smallest) set \( V_O \) of output orderings that is fair and complete.

On a side note, one can consider \( V_O \) as a set of “races” (i.e., total orderings) that is consistent with a pre-specified outcomes of “matches” (i.e., pairwise comparisons). This forms the basis of a game called “Match to Race” available online \(^5\).

The following example illustrates the setting.

**Example 1**: Let \( n = 3 \), i.e., there are three candidates \( a_1, a_2 \) and \( a_3 \). Let the set \( V_I \) consist of the following three orderings: \((a_1 > a_2), (a_2 > a_3 > a_1), (a_3 > a_1 > a_2)\). Thus, in the set of input orderings, \( a_1 \) beats \( a_2 \) and \( a_3 \) beats \( a_1 \) (there is no restriction on the comparison between \( a_2 \) and \( a_3 \)). For this example, the most efficient set \( V_O \)

\(^1\)This criterion of fairness is stronger than that considered by Arrow \(^1\).
comprises a single ordering \((a_3 > a_1 > a_2)\). Observe that \(V_O\) is complete since this ordering is a complete ordering on all \(n = 3\) candidates. It is also fair since \(a_1\) beats \(a_2\) and \(a_3\) beats \(a_2\) in \(V_O\). This \(V_O\) is the most efficient since there cannot be any other complete and fair output set that has a fewer (i.e., zero) number of orderings.

II. AGGREGATION ALGORITHM: UPPER BOUNDS ON THE SIZE OF \(V_O\)

We first present a lemma that will subsequently be employed as a tool for our main result. The proof of this lemma considers an existing set of output orderings for a subset of candidates and provides an algorithm for adding more candidates to this set while preserving fairness and completeness. The main result is presented subsequently in Theorem 2.

Lemma 1: Consider any positive integers \(m\) and \(u_m\). Suppose there exists an algorithm that can represent every set of input orderings on \(m\) candidates with a set of output orderings satisfying the fairness and completeness properties, such that the size of this set of output orderings is an odd number no larger than \(m\). Then one can represent every set of input orderings on \((m + 2)\) candidates with a set of output orderings satisfying the fairness and completeness properties, such that the size of this set of output orderings is no more than \((u_m + 2)\).

Proof: For any set of input orderings on \((m + 2)\) candidates, first pick any arbitrary subset of \(m\) candidates. In the set of input orderings, remove the remaining 2 candidates. Apply the given algorithm to this modified set of input orderings to obtain a set \(V_O\) consisting of at most \(u\) output orderings that is fair and complete on the set of \(m\) candidates. From the assumption of this lemma, \(u\) is odd and \(u \leq u_m\). Let \(a_{m+1}\) and \(a_{m+2}\) be the two additional candidates that have not been considered so far. We shall now insert the candidates \(a_{m+1}\) and \(a_{m+2}\), and in this process, add two additional orderings to the set \(V_O\). We shall ensure that this aggregate set of orderings is complete and fair with respect to these \((m + 2)\) candidates.

We need to introduce some notation at this point. If \(a_{m+1} > a_{m+2}\) in \(V_I\) then let \(b = a_{m+1}\) and \(c = a_{m+2}\); otherwise let \(b = a_{m+2}\) and \(c = a_{m+1}\). We first include the two new candidates in the existing \(u\) orderings in \(V_O\). To this end observe that \(u\) is an odd number. Choose \(\frac{u+1}{2}\) arbitrary orderings in \(V_O\) and insert “\(b > c >\)” at the top (i.e., making \(b\) the highest and \(c\) the next highest). In the remaining \(\frac{u-1}{2}\) orderings in \(V_O\), insert “\(> c > b\)” at the bottom (i.e., making \(b\) the lowest and \(c\) the next lowest). Call the resulting set of orderings as the ‘inherited’ set of orderings.

We now add two new orderings to \(V_O\). Partition the set \(\{a_1, \ldots, a_m\}\) into four subsets \((S_1, S_2, S_3, S_4)\) as follows

\[
\begin{align*}
S_1 &= \{a_j | a_j > b, a_j > c, j \in [m]\} \\
S_2 &= \{a_j | a_j > b, c \geq a_j, j \in [m]\} \\
S_3 &= \{a_j | b \geq a_j, a_j > c, j \in [m]\} \\
S_4 &= \{a_j | b \geq a_j, c \geq a_j, j \in [m]\}
\end{align*}
\]

For \(i \in \{1, 2, 3, 4\}\), let \(\bar{S}_i\) be any arbitrary ordering of the elements in \(S_i\), and let \(\bar{S}_i'\) be the reverse of \(\bar{S}_i\). Add the two following orderings to \(V_O\):

- “Penultimate ordering”: \(\bar{S}_2 > \bar{S}_1 > b > \bar{S}_3 > c > \bar{S}_4\)
- “Last ordering”: \(\bar{S}_4' > \bar{S}_3' > \bar{S}_1' > c > \bar{S}_2' > b\).

This completes the construction of \(V_O\) for the \((m + 2)\) candidates. The number of orderings in \(V_O\) is \((u + 2)\), which is no larger than \((u_m + 2)\).

We shall now show that this set of output orderings on the \((m + 2)\) candidates is complete and fair. Completeness is obvious since each ordering involves all \((m + 2)\) candidates \(\{a_1, \ldots, a_{m+2}\}\). We now show fairness. For any \((i, j) \in [m]^2\), the relationship between \(a_i\) and \(a_j\) is not altered in the set of inherited orderings. Among the two new orderings, one has \(a_i > a_j\) and the other has \(a_j > a_i\) and hence their relationship is retained. The fairness among comparisons involving \(b\) or \(c\) is established by counting the orderings in the following table which depicts the number of orderings supporting different pairwise comparisons. In the table, \(x_i\) is an arbitrary element of \(S_i\), for \(i \in \{1, 2, 3, 4\}\).
The following theorem presents the main result.

**Theorem 2:** For any $V_I$ on $n$ candidates, there exists a $V_O$ satisfying the fairness and completeness properties, with the number of elements in $V_O$ being at most

$$n - \lceil \log_2 (n + 1) \rceil + 1 \quad \text{if } n - \lceil \log_2 (n + 1) \rceil \text{ is even},$$

$$n - \lceil \log_2 (n + 1) \rceil + 2 \quad \text{if } n - \lceil \log_2 (n + 1) \rceil \text{ is odd}.$$

**Proof:** Let us represent the input $V_I$ as a directed graph: the graph has $n$ vertices representing the $n$ candidates, and for any $(i, j) \in [n]^2$, there is a directed edge from candidate $i$ to candidate $j$ if candidate $i$ beats candidate $j$ in the set of input orderings $V_I$. If there is no edge between $i$ and $j$ then add an edge between them with an arbitrary direction.

We now claim that this graph must have a subgraph of $\lceil \log_2 (n + 1) \rceil$ vertices that forms a directed acyclic graph, and the following algorithm finds that subgraph. First choose the vertex with the largest number of outgoing edges. Since the total number of vertices is $n$ and the total number of edges in the graph is $\frac{n(n-1)}{2}$, this vertex must have at least $\lceil \frac{n-1}{2} \rceil$ outgoing edges. Next, restrict your attention to the subgraph formed by the set of vertices which these edges are incident on. This subgraph will have a total of $\frac{n_1(n_1-1)}{2}$ edges. Choose the vertex with the largest number of outgoing edges in this subgraph; this number must be at least $\lceil \frac{n_1-1}{2} \rceil$. Next, again restrict your attention further to the subgraph formed by the set of vertices that form the destinations of these edges, and recurse until there are no more vertices left. By construction, this collection of these chosen vertices forms a directed acyclic graph. Let us now find a lower bound on the number of vertices in this directed acyclic graph. Let $f(m)$ be the size of the directed acyclic graph constructed in this manner, when there are $m$ vertices in the original graph. Clearly, $f(1) = 1$. From the arguments above, it must also be that

$$f(n) \geq 1 + f \left( \left\lceil \frac{n-1}{2} \right\rceil \right).$$

We now show, via an induction argument, that $f(n) \geq \lceil \log_2 (n + 1) \rceil$. This is trivially true for $n = 1$. For any higher value of $n$, suppose it is true for values smaller than $n$. Then, one can verify that

$$\lceil \log_2 (n + 1) \rceil \geq 1 + \log_2 \left( \left\lceil \frac{n-1}{2} \right\rceil + 1 \right),$$

which proves our claim.

We shall now construct a set of output orderings $V_O$ using the directed acyclic graph constructed above, and applying the procedure of Lemma 1 recursively to it. First, construct a single ordering on the set of candidates that form the directed acyclic graph: this ordering is the total ordering of the directed acyclic graph (i.e., the order in which the vertices were chosen in the procedure above). Now, from the remaining set of $(n - \lceil \log_2 (n + 1) \rceil)$ candidates, add two candidates at a time using the recursion of Lemma 1. Every time this recursion inserts two

<table>
<thead>
<tr>
<th>Pair\Orderings</th>
<th>Inherited</th>
<th>Pen.</th>
<th>Last</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b &gt; x_1$</td>
<td>$\frac{u_m+1}{2}$</td>
<td>0</td>
<td>0</td>
<td>$(u_m+1)/2$</td>
</tr>
<tr>
<td>$x_1 &gt; b$</td>
<td>$\frac{u_m-1}{2}$</td>
<td>1</td>
<td>1</td>
<td>$(u_m+3)/2$</td>
</tr>
<tr>
<td>$b &gt; x_2$</td>
<td>$\frac{u_m+1}{2}$</td>
<td>0</td>
<td>0</td>
<td>$(u_m+1)/2$</td>
</tr>
<tr>
<td>$x_2 &gt; b$</td>
<td>$\frac{u_m-1}{2}$</td>
<td>1</td>
<td>1</td>
<td>$(u_m+3)/2$</td>
</tr>
<tr>
<td>$b &gt; x_3$</td>
<td>$\frac{u_m+1}{2}$</td>
<td>1</td>
<td>0</td>
<td>$(u_m+3)/2$</td>
</tr>
<tr>
<td>$x_3 &gt; b$</td>
<td>$\frac{u_m-1}{2}$</td>
<td>0</td>
<td>1</td>
<td>$(u_m+1)/2$</td>
</tr>
<tr>
<td>$b &gt; x_4$</td>
<td>$\frac{u_m+1}{2}$</td>
<td>0</td>
<td>1</td>
<td>$(u_m+3)/2$</td>
</tr>
<tr>
<td>$x_4 &gt; b$</td>
<td>$\frac{u_m-1}{2}$</td>
<td>0</td>
<td>1</td>
<td>$(u_m+1)/2$</td>
</tr>
<tr>
<td>$b &gt; c$</td>
<td>$\frac{u_m+1}{2}$</td>
<td>1</td>
<td>0</td>
<td>$(u_m+3)/2$</td>
</tr>
</tbody>
</table>
additional candidates, it adds two additional orderings. Since we started with a single ordering, the total number of orderings always remains odd. Furthermore, after every recursion step, the set of orderings remains fair and complete in the set of candidates considered till then, thus satisfying the requirements of Lemma [1] Finally, if \( n - \lceil \log_2 (n + 1) \rceil \) is an odd number, then one candidate will remain to be included at the end of these steps. In order to accommodate this candidate, consider an additional fictitious candidate (with arbitrary positions in the orderings in \( V_I \)) and add these two candidates using Lemma [1] Then, simply discard the fictitious candidate from the set of output orderings. This results in a complete and fair set of output orderings with the size of this set as claimed in the statement of the Lemma.

II. LOWER BOUNDS ON THE SIZE OF \( V_O \)

The following simple proposition provides a lower bound on the size of \( V_O \).

**Proposition 3:** For any value of \( n \), there exists some input \( V_I \) on \( n \) candidates such that the number of orderings in any set \( V_O \) satisfying the fairness and completeness properties is at least

\[
\frac{n}{2 \log_2 (n + 1)}.
\]

**Proof:** Consider the non-trivial case of \( n > 1 \). The output orderings must specify the outcomes of comparisons between each of the \( \frac{n(n-1)}{2} \) pairs of candidates. Thus, one must obtain \( \frac{n(n-1)}{2} \) bits of information from the output orderings. Now, each output ordering is one out of \( n! \) complete orderings, and hence contributes \( \log_2 (n!) \) bits. It follows that the number of elements in \( V_O \) is at least

\[
\frac{n(n-1)}{2 \log_2 (n!)}. 
\]

Applying the bound \( n! \leq (n + 1)^{n-1} \) gives the desired result.

IV. DISCUSSION

The lower bounds (Theorem [2]) on the number of elements in \( V_O \) are a logarithmic factor away from the upper bounds (Proposition [3]). Reducing this gap is an open problem.

The algorithms for constructing \( V_O \) derived in this writeup also forms the basis of a new game called ‘Match to Race’ available online [5]. The game requires the user to convert a given set of pairwise matches into a set of orderings of all candidates, and uses a slightly weaker version of the algorithm of Theorem [2] for its operations.

REFERENCES