Updates on Global Ranking Problem

Contents

1 Problem Definition

The problem is formulated with the following assumptions in mind. Note that this formulation applies to a general setting where pairwise comparisons can be performed on some set of items. Depth perception is one example of a possible setting, but we are still to determine if it is a good fit for the problem.

1.1 Preliminaries

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be the set of all *items*, and $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ be the set of all *workers*.

Definition 1 (Query)**.** A *query* is a subset of items in $\mathcal{X}$ which is allocated to some worker(s). Formally, we define a set of queries, $\mathcal{Q} = \{q_1, q_2, \ldots, q_r\}$ where $\forall\, i \in \{1, 2, \ldots, r\}$, $q_i \subseteq \mathcal{X}$. A worker gives pair-wise comparisons on items within an allocated query which will be used to generate a *sort order*, defined below.[1]

Definition 2 (Query-Response Graph)**.** The *query-response graph* $G_{q,w}(V, E)$ is a graphical representation of the responses given by a worker w for a query q, where there is a vertex $v_i \in V$ corresponding to every item x_i in q, and an edge $e_{ij} \in E$ from v_i to v_j if $x_i < x_j$ holds $(x_i, x_j \in q)$.

Definition 3 (Operator $\odot$)**.** The operation $(x_i \odot x_j)$ denotes a pair-wise comparison between $x_i, x_j \in q$ for some query q.

Definition 4 (Sort Order)**.** A *sort order*, $\phi(q, w)$ is an ordering generated by worker w for query q. The set of all sort orders is denoted by $\mathcal{S}$. $\phi(q, w)$ is a sort order if and only if,

(1) The query-response graph $G_{q,w}$ is a directed acyclic graph (DAG).

(2) The pair-wise comparison $(x_i \odot x_j)$ is known $\forall\, i, j \in q$, $i \neq j$. Equivalently, the underlying, undirected graph of $G_{q,w}$ must be completely connected.[2]

(1) ensures that each worker's comparisons are internally consistent for a query. (1) and (2) together ensure that the set of pairwise-comparisons induces a unique ordering on items.[3]

1.2 Worker Perceptions and Compatibility

Definition 5 (Underlying Perception)**.** An *underlying perception* is a worker's inherent way of ordering some items, *i.e.* a worker generates $\phi(q, w)_{|\mathrm{P}}$ (read as $\phi(q, w)$ adheres to P) for a query q in accordance with a perception $\mathrm{P} \in \mathcal{P}$ the set of all perceptions. We may omit the subscript $_{|\mathrm{P}}$ for brevity, noting that $\phi(q, w)$ essentially implies $\phi(q, w)_{|\mathrm{P}}$ for some P. Note that $\mathcal{P}$ is not directly knowable, but is expressed through the various $\phi(q, w)$ sort orders on different queries solved by workers.

[1] Assume for now that the composition of all queries is pre-decided by some sampling algorithm. This sampling algorithm should ideally attempt to minimize the number of queries needed to discover all worker perceptions.

[2] Note that, for (2) to hold, we do not necessarily need to ask the worker for all $\binom{|q|}{2}$ pairwise comparisons. For instance, let $q = \{1, 2, 3\}$ be the query. Only having the worker tell us $1 < 2$ and $2 < 3$ is sufficient for (2) to hold, since we can necessarily infer $1 < 3$. Thus, we must either know, or can infer the answer to every comparison possible in q.

[3] Assume the existence of a query-management algorithm. Given a query q, the algorithm asks the worker w for answers to $(x_i \odot x_j)$ pair-wise comparisons sequentially. No pair-wise comparison is repeated, and the algorithm attempts to minimize the number of comparisons asked in order to generate a $\phi(q, w)$ for the query.

Our goal is to solve the following problem,

> Find a $\mathcal{P}$ of minimum size such that every $\phi(q,w)$ given to us adheres to some $\mathrm{P} \in \mathcal{P}$ *i.e.* it can be written as $\phi(q,w)_{|\mathrm{P}}$ for some $\mathrm{P} \in \mathcal{P}$.

Note that this problem is not directly solvable, since it is impossible to directly observe or manipulate underlying perceptions. However, since $\phi(q,w)$ has been derived from the worker's underlying perception, it can be directly used as a proxy for his/her underlying perception.

Definition 6 (Consistency/Compatibility)**.** Informally, we say that two sort orders are *consistent* (or *compatible*) if the ordering on every pair of items common to both sort orders is identical. Note that consistency is not affected by items that are non-overlapping; these items can be ordered arbitrarily without affecting consistency.

Suppose we have two workers, w_1 and w_2, who have given us sort orders on different queries, $\phi(q_1,w_1)_{|\mathrm{P}_1}$ and $\phi(q_2,w_2)_{|\mathrm{P}_2}$. We say that $\phi(q_1,w_1)_{|\mathrm{P}_1}$ is consistent with $\phi(q_2,w_2)_{|\mathrm{P}_2}$ if $\forall\, x_i, x_j \in q_1 \bigcap q_2$, the response to $(x_i \odot x_j)_{w_1} = (x_i \odot x_j)_{w_2}$. When this occurs, we say that $\mathrm{P}_1 = \mathrm{P}_2$ for $\phi(q_1,w_1)_{|\mathrm{P}_1}$ & $\phi(q_2,w_2)_{|\mathrm{P}_2}$.

Thus, given sort orders for some queries, the notion of consistency/compatibility allows us to check which sort orders adhere to the same perception.

1.3 Worker Response Aggregation and Rankings

Definition 7 (Aggregate-Response Graph)**.** The *aggregate-response graph*, $G_{\mathbf{S}}(V,E)$ is defined over a set $\mathbf{S} = \{\phi_1(q_1,w_1),\ldots,\phi_l(q_l,w_l)\}$ *i.e.* a set of sort orders for different queries. Intuitively, $G_{\mathbf{S}}$ is the superposition of the query-response graphs $\{G_{q_1,w_1},\ldots,G_{q_l,w_l}\}$. $G_{\mathbf{S}}$ has a vertex $v_i \in V$ corresponding to every item $x_i \in q_1 \bigcup \cdots \bigcup q_l$. There is an edge $e_{i,j} \in E$ from v_i to v_j if $x_i < x_j$ holds in *any* sort order in $\mathbf{S}$.

Definition 8 (Ranking)**.** Informally, a *ranking* is an aggregation of compatible sort orders, *i.e.* sort orders that share the same underlying perception. Formally, a ranking R_i is an aggregation of all sort orders of the form $\phi(q_k,w_k)_{|\mathrm{P_i}}$, *i.e.* all sort orders that adhere to $\mathrm{P_i}$.

WLOG, let the set of sort orders that adhere to $\mathrm{P_i}$ be $\mathbf{S} = \{\phi_1(q_1,w_1),\ldots,\phi_l(q_l,w_l)\}$. A ranking R_i can be inferred from the aggregate-response graph $G_{\mathbf{S}}$ *if and only if* $G_{\mathbf{S}}$ is a DAG.[4,5] We will say that each $\phi_k(q_k,w_k) \in \mathbf{S}$ is *compatible* with R_i.

WLOG, we will assume for the rest of our discussion below that each worker answers exactly 1 query, *i.e.* $n = m$. Thus, we have $\mathcal{S} = \{\phi_1(q_1,w_1),\ldots,\phi_n(q_n,w_n)\}$.

Definition 9 (Worker Graph)**.** Intuitively, the *worker graph*, $G_{\mathcal{W}}$ represents compatibility between pairs of workers. Each vertex $v_i \in V$ corresponds to one worker w_i, and an edge $e_{ij} \in E$ between vertex v_i and v_j exists if $\phi_i(q_i,w_i)$ is compatible with $\phi_j(q_j,w_j)$.

[4]The uniqueness of R_i depends on whether $G_{\mathbf{S}}$ has a single, unique topological ordering. If not, then R_i is not unique, and can be chosen as any valid topological ordering of $G_{\mathbf{S}}$.

[5]Note that it is possible for $G_{\mathbf{S}}$ to *not* be a DAG and contain one or more cycles. This case and its implications will be addressed in Section 2.3.

1.4 Problem Formulation

Given the preliminary outline above, we now formulate 2 concrete problems that we will solve in subsequent sections. We term these problems the Min-Multiple-Rankings problem, and the Min-Global-Rankings problem.

1.4.1 Min-Global-Rankings

Informally, we state the Min-Global-Rankings problem as,

> Find the minimum number of rankings, such that each sort order is consistent with at least one ranking, and each ranking is defined over the entire set of items $\mathcal{X}$.

Recall that we are given a set of sort orders $\mathcal{S} = \{\phi_1(q_1, w_1), \ldots, \phi_n(q_n, w_n)\}$, and let $\mathcal{R}$ be the complete set of permutations over *all items* in $\mathcal{X}$. Our problem is then stated as,

> Find $\mathbf{R} \subset \mathcal{R}$ of minimum size, such that $\phi_k(q_k, w_k)$ is consistent with some $R_i \in \mathbf{R}$, $\forall\ \phi_k(q_k, w_k) \in \mathcal{S}$.

1.4.2 Min-Multiple-Rankings

The Min-Multiple-Rankings problem is a relaxation of the Min-Global-Rankings problem, where we remove the constraint that every ranking must be defined over $\mathcal{X}$. Informally, we state the problem as,

> Find the minimum number of rankings, such that each sort order is consistent with at least one ranking.

Once again, we are given a set of sort orders $\mathcal{S} = \{\phi_1(q_1, w_1), \ldots, \phi_n(q_n, w_n)\}$, and let $\mathcal{R}$ be the complete set of permutations over the *power set* of $\mathcal{X}$. Our problem is then stated as,

> Find $\mathbf{R} \subset \mathcal{R}$ of minimum size, such that $\phi_k(q_k, w_k)$ is consistent with some $R_i \in \mathbf{R}$, $\forall\ \phi_k(q_k, w_k) \in \mathcal{S}$.

We now move on to proving NP-Hardness of Min-Multiple-Rankings and Min-Global-Rankings.

2 NP-Hardness of Ranking Problems

We first prove NP-Hardness of the Min-Multiple-Rankings problem. We propose a polynomial-time reduction from a general instance of Min-Clique-Cover to a general instance of Min-Multiple-Rankings. However, we note that it may not always be possible to obtain a solution to the Min-Multiple-Rankings from a solution of Min-Clique-Cover. Particularly, as we discuss in Section 2.3, cases commonly called the 'Condorcet Paradox' may occur, which do not readily provide a method to obtain a ranking. To circumvent this flaw, we outline an approach in 2.4 that allows us to construct a special sub-class of Min-Multiple-Rankings from general, hard instances of Min-Clique-Cover. We note that for this special sub-class of Min-Multiple-Rankings, the polynomial time reduction proposed in Section 2.2 is valid, and the existence of a ranking is guaranteed.

Consider the statement of Min-Multiple-Rankings given in Section 1.4.2. We can interpret this problem as finding a partition of the worker graph into compatible groups of workers (each compatible group corresponds to a clique in this graph). The aggregate-response graph for each compatible group must be a DAG, and can therefore be used to obtain a ranking. In subsequent subsections, we use this interpretation to create a reduction from Min-Clique-Cover.

2.1 Clique Cover Problem

The Clique-Cover problem is one of Richard Karp's original 21 problems [1] that were shown to be NP-Complete. A clique cover of $\mathcal{G}(V, E)$ is a family $\mathcal{F}$ of cliques such that for every $v \in V$, there exists some $S \in \mathcal{F}$ such that $v \in S$. The Min-Clique-Cover problem is formally defined as,

Determine a clique cover $\mathcal{F}$ of minimum size for a given graph $\mathcal{G}$.

The decision variant of Min-Clique-Cover (*i.e.* Clique-Cover) can be proven NP-Complete by reducing Graph K-Colorability to Clique-Cover.

2.2 Polynomial Time Reduction

Given an instance of Min-Clique-Cover on $\mathcal{G}(V, E)$, we create another graph, $\mathcal{G}'$, with identical edges and vertices. For $\mathcal{G}'$ to be a valid instance of Min-Multiple-Rankings, we must add a notion of workers and worker responses to $\mathcal{G}'$. We interpret $\mathcal{G}'$ as a worker graph (as defined in Section 1.3), where each vertex represents a worker, and edges represent pair-wise consistency.

Finding the Min-Clique-Cover corresponds to finding a clique partition of workers of minimum size. All workers in each clique are pair-wise compatible, and every worker lies in at-least one clique. A ranking can possibly be inferred from each clique if the aggregate-response graph corresponding to the clique is a DAG. In other words, we claim that a solution of Min-Multiple-Rankings can be used to obtain a solution of Min-Clique-Cover.

2.2.1 Proof of Correctness

Claim 1. A solution of Min-Clique-Cover corresponds to a valid solution of Min-Multiple-Rankings, and vice versa.

Proof. Since $\mathcal{G}$ and $\mathcal{G}'$ are identical, any clique cover of $\mathcal{G}$ is also a clique cover of $\mathcal{G}'$. Each clique in $\mathcal{G}'$ will correspond to a ranking if the aggregate-ranking graph corresponding to that clique is a DAG. To guarantee this, we show a construction in Section 2.4, and defer this discussion for later.

In the other direction, any ranking in $\mathcal{G}'$ will correspond to a clique, and since all workers in $\mathcal{G}'$ must belong to some ranking, these cliques will also partition $\mathcal{G}$. □

Claim 2. The minimum number of cliques that cover $\mathcal{G}$ is equal to k if and only if the minimum number of rankings in $\mathcal{G}'$ is also equal to k.

Proof. Given that the minimum number of rankings in $\mathcal{G}'$ is k, let the minimum number of cliques in $\mathcal{G}$ be l, where $l \neq k$. If $l > k$, then there must be some worker in $\mathcal{G}'$ that does not belong to any ranking, since the only reason that the minimum number of cliques in $\mathcal{G}$ is more than k is because we cannot cover $\mathcal{G}$ with k cliques. But since $\mathcal{G}$ is identical to $\mathcal{G}'$ there must be workers in $\mathcal{G}'$ that do not belong to any ranking. Therefore, the partition into k rankings will not cover all workers, which is a contradiction.

Now, let $\mathcal{R}$ be the minimum clique cover of $\mathcal{G}$ with size $l < k$. By partitioning $\mathcal{G}'$ into $\mathcal{R}$, we can find a smaller number of rankings than k. This contradicts the clause that the original partition of $\mathcal{G}'$ into k rankings is minimum.

Given that the minimum clique cover of $\mathcal{G}$ is of size k, let the smallest partition of $\mathcal{G}'$ into rankings be of size l, where $l \neq k$. Similar to the reasoning above, if $l > k$, then some vertices of $\mathcal{G}$ must have been excluded from the clique cover of $\mathcal{G}$, which would lead to a contradiction, since we were told that the partition over $\mathcal{G}$ is in fact a clique cover. Similarly, if $l < k$, then by reconstructing the clique cover in $\mathcal{G}$, we can get a clique cover with a smaller size, which is a contradiction. □

2.3 Rock-Paper-Scissors Instances in Min-Multiple-Rankings

Some instances of Min-Multiple-Rankings, suffer from what we call the Rock-Paper-Scissors problem. For these instances, the aggregate-response graph corresponding to some clique in the worker graph is not a DAG. Thus, we cannot find a ranking for this clique in the worker graph, and therefore cannot construct a solution for Min-Multiple-Rankings. For instance, consider a group of 3 pair-wise compatible workers, A, B and C, that give the following sort orders on their respective queries: $A \rightarrow \{\text{ROCK} < \text{PAPER}\}; B \rightarrow \{\text{PAPER} < \text{SCISSOR}\}$ and $C \rightarrow \{\text{SCISSOR} < \text{ROCK}\}$. If A, B and C belong to the same clique, the aggregate-response graph contains the cycle ROCK $\rightarrow$ PAPER $\rightarrow$ SCISSOR $\rightarrow$ ROCK and the clique as a whole is incompatible. This issue is frequently observed in elections scenarios, wherein the result of the election as a whole may be intransitive even though each voter specified a transitive preference. In literature, this issue is often called the *Voting paradox* or the *Condorcet paradox.*

For Min-Multiple-Rankings, the Condorcet paradox implies that a solution of Min-Clique-Cover may not allow us to construct a solution of Min-Multiple-Rankings. To circumvent this issue, we propose a construction in the subsequent section that allows us to reduce a general, hard instance of Min-Clique-Cover to a special subclass of Min-Multiple-Rankings.

2.4 Construction of Special Subclass of Min-Multiple-Rankings

In this section, we propose an algorithm that converts an instance of Min-Clique-Cover to a valid instance of Min-Multiple-Rankings. For this conversion to generate a valid instance of Min-Multiple-

Rankings, every clique in the worker graph must have a corresponding aggregate-response graph that is a DAG. This guarantees that we can construct a ranking for each clique.

Keeping these objectives in mind, the construction is shown in Algorithm 1.

Algorithm 1 Min-Multiple-Rankings Instance Construction

```
1: procedure ConstructGraph($\mathcal{G}$)
2:     AssignedSamples = []                    ▷ Dictionary of discovered nodes and assigned samples
3:     FinalSamples = []                       ▷ Look-up Dictionary of all nodes and assigned samples
4:     Counter = 1                             ▷ Number of items assigned to samples yet
5:     for $v_i \in V$ do
6:         IncompatibleNodes$_i \leftarrow \{v_j : Edge(v_i, v_j) \notin E; i \neq j\}$   ▷ All nodes incompatible with $v_i$
7:         SizeOfSample$_i \leftarrow 2 \times |$IncompatibleNodes$_i|$   ▷ Size of sample assigned to $v_i$
8:         if |IncompatibleNodes$_i$ == 0| then   ▷ Node compatible with all other nodes
9:             SizeOfSample$_i \leftarrow 2$
10:        end if
11:        for $x_j \in$ IncompatibleNodes$_i$ do
12:            if $x_j \in$ AssignedSamples.$keys$ then   ▷ Find common items with incompatible nodes
13:                NewPair $\leftarrow$ $GetPairOfSamples$(AssignedSamples, $x_j$))
14:                AssignedSamples[$v_i$].$append$(NewPair)
15:                FinalSamples[$v_i$].$append$(NewPair)
16:                SizeOfSample$_i \leftarrow$ SizeOfSample$_i - 2$
17:            end if
18:        end for
19:        for $i \leftarrow \{1, \ldots,$ SizeOfSample$_i\}$ do
20:            AssignedSamples[$v_i$].$append$(counter)   ▷ Add all remaining items
21:            FinalSamples[$v_i$].$append$(counter)
22:            counter $\leftarrow$ counter + 1
23:        end for
24:    end for
25: return
26: end procedure
27:
28: procedure GetPairOfSamples(AssignedSamples, $x_j$)   ▷ Returns a pair of items for a sample
29:     item$_1$ = AssignedSamples[$x_j$].$pop$()
30:     item$_2$ = AssignedSamples[$x_j$].$pop$()
31: return (item$_1$, item$_2$)
32: end procedure
```

This algorithm is demonstrated step wise on an instance of Min-Clique-Cover, as shown in Figure 1.

- **Step 1:** From the given graph $\mathcal{G}$, an identical graph $\mathcal{G}'$ is constructed. $\mathcal{G}'$ is interpreted as a worker graph as shown in Figure 2. Finding a set of rankings over cliques in $\mathcal{G}'$ is an instance of Min-Multiple-Rankings.

- **Step 2:** We start with w_1. Since w_1 is incompatible with *3* other vertices (w_5, w_6, w_7), the query q_1 answered by w_1 will be over $2 \times 3 = 6$ items. WLOG, let $q_1 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. At this stage, our book-keeping table is shown in Table 1.

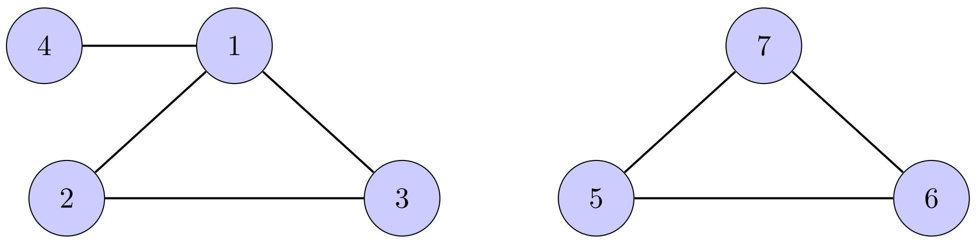

Figure 1: Graph $\mathcal{G}$, an instance of Min-Clique-Cover.

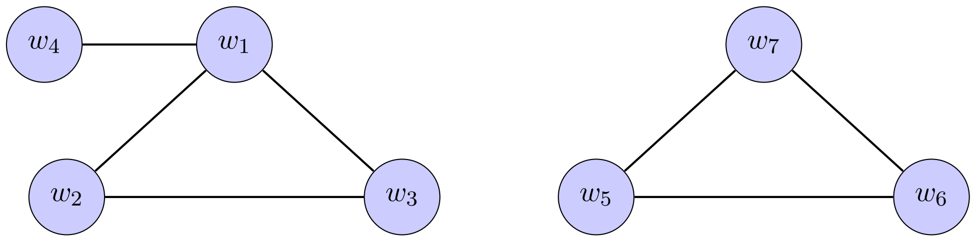

Figure 2: Graph $\mathcal{G}'$, an instance of Min-Multiple-Rankings.

Worker	*Assigned Samples*
w_1	$x_1, x_2, x_3, x_4, x_5, x_6$

Table 1: Book-keeping Table after **Step 2**.

- **Step 3:** For w_2, the number of inconsistent vertices is 4 (w_4, w_5, w_6, w_7). Therefore, q_2 answered by w_2 comprises of $2 \times 4 = 8$ items. Since w_2 has an edge with w_1, therefore there should be no overlap between q_1 and q_2, to ensure compatibility, i.e. $q_1 \bigcap q_2 = \phi$. Again, WLOG let $q_2 = \{x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}\}$. The state of our book-keeping table is shown in Table 2.

Worker	*Assigned Samples*
w_1	$x_1, x_2, x_3, x_4, x_5, x_6$
w_2	$x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}$

Table 2: Book-keeping Table after **Step 3**.

- **Step 4:** w_3 is inconsistent with the same 4 vertices as w_2, and therefore q_3 answered by w_3 contains 8 elements. Also, w_3 has edges with both discovered nodes (w_1 and w_2), and so $q_3 \bigcap q_1 = \phi$ and $q_3 \bigcap q_2 = \phi$. $q_3 = \{x_{15}, x_{16}, \ldots, x_{22}\}$, and the state of the book-keeping table is shown in Table 3.

Worker	*Assigned Samples*
w_1	$x_1, x_2, x_3, x_4, x_5, x_6$
w_2	$x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}$
w_3	$x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}, x_{22}$

Table 3: Book-keeping Table after **Step 4**.

- **Step 5:** w_4 is inconsistent with $(w_2, w_3, w_5, w_6, w_7)$, therefore q_4 should have $2 \times 5 = 10$ items. However, w_4 is inconsistent with two workers that have already been discovered, and therefore must rank at least a single pair of items each from q_2 and q_3. Therefore, q_4 must contain x_7, x_8 (from

q_2) and x_{15}, x_{16} (from q_3). Besides these 4 items, q_4 must also contain 6 other items, as a result of which $q_4 = \{x_7, x_8, x_{15}, x_{16}, x_{23}, x_{24}, x_{25}, x_{26}, x_{27}, x_{28}\}$. It is important to note that $q_4 \bigcap q_1 = \phi$ (ensuring compatibility), $q_4 \bigcap q_2 = \{x_7, x_8\}$ and $q_4 \bigcap q_2 = \{x_{15}, x_{16}\}$. Further, the response of w_4 to ($x_7 \odot x_8$) must be the opposite of the response given by w_2 for the same comparison (to ensure incompatibility between w_2 and w_4). For example, if w_2 gave $x_7 < x_8$, then w_4 must give $x_7 > x_8$. This holds for responses over $\{x_{15}, x_{16}\}$ as well. The book-keeping table is shown in Table 4.

Worker	*Assigned Samples*
w_1	$x_1, x_2, x_3, x_4, x_5, x_6$
w_2	~~x_7, x_8~~, $x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}$
w_3	~~x_{15}, x_{16}~~, $x_{17}, x_{18}, x_{19}, x_{20}, x_{21}, x_{22}$
w_4	~~(x_7, x_8)~~$_{Flipped}$, ~~(x_{15}, x_{16})~~$_{Flipped}$, $x_{23}, x_{24}, x_{25}, x_{26}, x_{27}, x_{28}$

Table 4: Book-keeping Table after **Step 5**.

- **Step 6:** w_5 is inconsistent with w_1, w_2, w_3, w_4, and so q_5 has 8 items. Since all of these nodes have already been discovered, w_5 must rank a pair of items each from q_1, q_2, q_3 and q_4. As a result, $q_5 = \{x_1, x_2, x_9, x_{10}, x_{17}, x_{18}, x_{23}, x_{24}\}$. Note that $q_5 \bigcap q_1 = \{x_1, x_2\}$ and the response given by w_5 to ($x_1 \odot x_2$) is the opposite of the response given by w_1, which ensures incompatibility between w_1 and w_5. Similarly, incompatibility can be ensured between w_5 and each of w_2, w_3 and w_4. The book-keeping table is shown in Table 5.

Worker	*Available Nodes*
w_1	~~x_1, x_2~~, x_3, x_4, x_5, x_6
w_2	~~x_7, x_8~~, ~~x_9, x_{10}~~, $x_{11}, x_{12}, x_{13}, x_{14}$
w_3	~~x_{15}, x_{16}~~, ~~x_{17}, x_{18}~~, $x_{19}, x_{20}, x_{21}, x_{22}$
w_4	~~(x_7, x_8)~~$_{Flipped}$, ~~(x_{15}, x_{16})~~$_{Flipped}$, ~~x_{23}, x_{24}~~, $x_{25}, x_{26}, x_{27}, x_{28}$
w_5	~~(x_1, x_2)~~$_{Flipped}$, ~~(x_9, x_{10})~~$_{Flipped}$, ~~(x_{17}, x_{18})~~$_{Flipped}$, ~~(x_{23}, x_{24})~~$_{Flipped}$

Table 5: Book-keeping Table after **Step 6**.

- **Step 7:** Similar to w_5, w_6 is inconsistent with w_1, w_2, w_3, w_4, and so must have 8 items in q_6. q_6 must consist of $\{x_3, x_4, x_{11}, x_{12}, x_{19}, x_{20}, x_{25}, x_{26}\}$. Note that $q_6 \bigcap q_5 = \phi$, which ensures compatibility with w_5. Furthermore, $q_6 \bigcap q_1 = \{x_3, x_4\}$, while the responses to ($x_3 \odot x_4$) given by w_6 and w_1 oppose each other, ensuring incompatibility between them. Similarly, incompatibility is ensured between w_6 and each of w_2, w_3 and w_4. The book-keeping table is shown in Table 6.

Worker	*Assigned Samples*
w_1	~~x_1, x_2~~, ~~x_3, x_4~~, x_5, x_6
w_2	~~x_7, x_8~~, ~~x_9, x_{10}~~, ~~x_{11}, x_{12}~~, x_{13}, x_{14}
w_3	~~x_{15}, x_{16}~~, ~~x_{17}, x_{18}~~, ~~x_{19}, x_{20}~~, x_{21}, x_{22}
w_4	~~(x_7, x_8)~~$_{Flipped}$, ~~(x_{15}, x_{16})~~$_{Flipped}$, ~~x_{23}, x_{24}~~, ~~x_{25}, x_{26}~~, x_{27}, x_{28}
w_5	~~(x_1, x_2)~~$_{Flipped}$, ~~(x_9, x_{10})~~$_{Flipped}$, ~~(x_{17}, x_{18})~~$_{Flipped}$, ~~(x_{23}, x_{24})~~$_{Flipped}$
w_6	~~(x_3, x_4)~~$_{Flipped}$, ~~(x_{11}, x_{12})~~$_{Flipped}$, ~~(x_{19}, x_{20})~~$_{Flipped}$, ~~(x_{25}, x_{26})~~$_{Flipped}$

Table 6: Book-keeping Table after **Step 7**.

- **Step 8:** Similar to w_5 and w_6, w_7 is incompatible with w_1, w_2, w_3, w_4, requiring q_7 to be of size 8. Same as in w_5 and w_6, each pair of items in q_7 will be common with one of w_1, w_2, w_3 or w_4. We construct $q_7 = \{x_5, x_6, x_{13}, x_{14}, x_{21}, x_{22}, x_{27}, x_{28}\}$. $q_7 \bigcap q_5 = \phi$ and $q_7 \bigcap q_6 = \phi$, to ensure compatibility between the three workers. Further, as shown in the case of w_5 and w_6, w_7 is incompatible with w_1, w_2, w_3 and w_4. The final book-keeping table is shown in Table 7.

Worker	*Assigned Samples*
w_1	~~x_1, x_2~~, ~~x_3, x_4~~, ~~x_5, x_6~~
w_2	~~x_7, x_8~~, ~~x_9, x_{10}~~, ~~x_{11}, x_{12}~~, ~~x_{13}, x_{14}~~
w_3	~~x_{15}, x_{16}~~, ~~x_{17}, x_{18}~~, ~~x_{19}, x_{20}~~, ~~x_{21}, x_{22}~~
w_4	~~(x_7, x_8)~~$_{Flipped}$, ~~(x_{15}, x_{16})~~$_{Flipped}$, ~~x_{23}, x_{24}~~, ~~x_{25}, x_{26}~~, ~~x_{27}, x_{28}~~
w_5	~~(x_1, x_2)~~$_{Flipped}$, ~~(x_9, x_{10})~~$_{Flipped}$, ~~(x_{17}, x_{18})~~$_{Flipped}$, ~~(x_{23}, x_{24})~~$_{Flipped}$
w_6	~~(x_3, x_4)~~$_{Flipped}$, ~~(x_{11}, x_{12})~~$_{Flipped}$, ~~(x_{19}, x_{20})~~$_{Flipped}$, ~~(x_{25}, x_{26})~~$_{Flipped}$
w_7	~~(x_5, x_6)~~$_{Flipped}$, ~~(x_{13}, x_{14})~~$_{Flipped}$, ~~(x_{21}, x_{22})~~$_{Flipped}$, ~~(x_{27}, x_{28})~~$_{Flipped}$

Table 7: Book-keeping Table after **Step 8**.

Notice that by this construction, the aggregate-ranking graph for each clique is always a DAG (since the set of items is non-overlapping), which guarantees that a ranking exists for each clique.

For each vertex v_i in the graph, we add $|V| - 1 - d_i$ new items over which the sorting is performed. $|V|$ stands for the total number of vertices in the graph, and d_i stands for the degree of v_i. Therefore, the total number of items for the entire graph will be

$$|\mathcal{X}| = \sum_{i=1}^{|V|} |V| - 1 - d_i = |V|^2 - |V| - 2|E|$$

As in the example demonstrated, where there were 7 vertices and 7 edges, we end up getting a list of $7^2 - 7 - 2 \times 7 = 28$ items.

2.4.1 Construction Invariants

Below, we list the invariants that hold at each iteration of the algorithm.

1. If two workers w_i and w_j are compatible with each other, then the set of items sorted by w_i is disjoint with the set of items sorted by w_j *i.e.* $q_i \bigcap q_j = \phi$.
2. Thus, for any clique of workers, each worker ranks a completely different set of items, i.e. $q_i \bigcap q_j = \phi$ for every w_i and w_j in the clique. This ensures that for a pair-wise compatible clique of workers, it will always be possible to obtain a ranking from their sort orders, without any ROCK-PAPER-SCISSORS cases.
3. If two workers w_i and w_j are incompatible, then there is a pair of items common to both q_i and q_j. The response given by w_j on this common pair is opposite to that given by w_i. This ensures incompatibility between any two workers.

3 Greedy Heuristic for Finding Minimum True Global Sort Orders

Given this setting, we can propose the following heuristic to find the minimum number of true global sort orders:

1. Begin with an empty list of global sort orders, L, and add the (empty) first ordering, L_1 to it.
2. Pick some worker w_i, who was queried some subset of nodes Q_j. Since w_i's perception is consistent with itself, we can add the order obtained from Q_j to L_1. This order is found in the following manner:
   (1) Find a sort order for the DAG provided by w_i over Q_j. There will exist a unique ordering, since the underlying skeleton graph (inferred from all pairwise comparisons provided by the worker) is a clique.
   (2) Add this order to L_1.
3. Pick the next worker w_{i+1}, who was queried the set Q_{j+1}, and repeat Step 2, unless a differing worldview is observed. A differing worldview is observed when, for two nodes X_a and X_b,
   (a) Both nodes X_a and X_b lie in L_1, and also exist in Q_{j+1} queried to worker w_{i+1}.
   (b) The ordering present in L_1 and as given by w_{i+1} (over Q_{j+1}) are opposite.

   In case a clash is observed, a new ordering L_2 has to be created, to which all elements of Q_{j+1} are added. [6] The new ordering L_2 that's been created will also contain all worker responses from the previous ordering that don't clash with w_{i+1}'s perception. [7]
4. Repeat Step 3 for the remaining queries, creating a new list L_k whenever a clash is obtained with elements in each list $\{L_1, L_2, ..., L_{k-1}\}$ belonging to L. In case the new query Q_t has no clashes with a subset of lists L' in L, then the order extracted from Q_t is added to every list in L', and no new list is created.

3.1 Complexity Analysis

For each query answered by some worker, it takes $O(n \log n)$ time to find their sort order. While checking for clashes for any query Q_i, it takes $O(n)$ time to check if a clash exists in any one list in L. This checking needs to be performed for all lists in L (at most w), and for each worker response (at most w, again). Therefore, the upper bound on the time complexity is $O(wn(w + \log n))$.

[6] We add all elements of Q_{j+1} to L_2, and not only the elements that are responsible for the clash, because of identifying the requirement of an entirely new perception, as observed by the w_{i+1}.

[7] Let the clash encountered between L_1 and q_{i+1} be between two nodes X_a and X_b. It is possible that when the edge between X_a and X_b was added to L_1, it was not directly inferred from a single worker's response to a query, but inferred over the responses of multiple workers (over multiple queries). In the example in Figure 3 (appearing in a later section), two worker responses of the form $(1 < 2 < 3)$ and $(2 < 3 < 4)$ also entail the existence of the edge $(1 < 4)$, even though nodes '1' and '4' belong to two different worker cliques. In this case, if the clash is between nodes '1' and '4' appearing in L_1 and worker w_{i+1}'s response, then responses from both workers will not be added to L_2, whilst all other workers compatible with w_{i+1}'s response will be added to L_2.

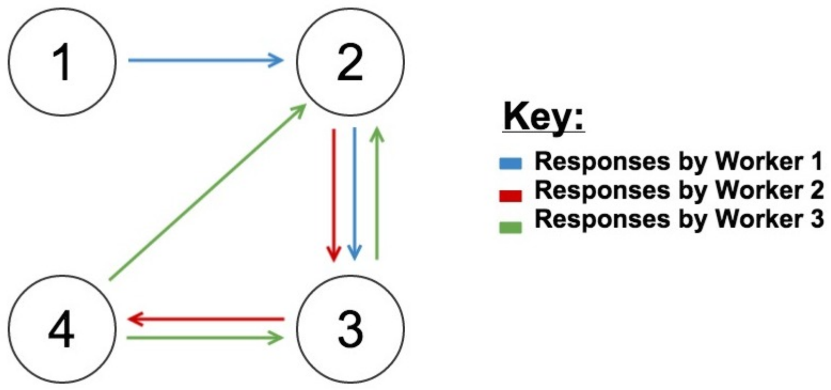

Figure 3: This figure demonstrates a possible failure case of the graph, as discussed in the text. Worker 1 was queried the subset $Q_1 := \{1, 2, 3\}$, while worker 2 and worker 3 were both queried $Q_2 := \{2, 3, 4\}$.

Now, if we can assume each query to have a maximum possible size $m = f(n)$, then the time complexity of the algorithm becomes $O(wm(w + \log m))$. Thus, the time complexity of this algorithm is weakly dependent on the overall number of items that must be sorted.

3.2 Failure Cases for Proposed Algorithm

In this section, we discuss some cases that cannot be handled by the algorithm provided. These cases arise when we find the global sort orders in a post-hoc manner, i.e. having collected worker responses over the set Q using some sampling strategy, we find sort orders as a post-processing step. An adaptive sampling methodology could help mitigate these issues.

Figure 3 discusses one of these failure cases. The set $\mathcal{S}$ consist of elements $\{1, 2, 3, 4\}$. There are two queries:

$$Q_1 = \{1, 2, 3\}$$
$$Q_2 = \{2, 3, 4\}$$

These are queried by three workers, w_1, w_2 and w_3. Query Q_1 is queried only to w_1, while Q_2 is queried to both workers w_2 and w_3. The answers given by workers w_1, w_2 and w_3 are:

$$w_1 \rightarrow 1 < 2 < 3$$
$$w_2 \rightarrow 2 < 3 < 4$$
$$w_3 \rightarrow 3 < 4 < 2$$

Given this example, one evident global ordering is $1 < 2 < 3 < 4$, which is aggregated from the perceptions of workers w_1 and w_2. However, since we've been given the responses of worker w_3 as well, we are aware that an alternative perception of the ordering exists, but it is impossible to extrapolate w_3's ordering over all elements of $\mathcal{S}$, for the following reasons:

- To create a cohesive global ordering from the responses of w_3, we need to have a sense of where to place node '1' in w_3's ordering of $3 < 4 < 2$. However, the only pairwise comparison we have that includes '1' has been provided by w_1, which ideally should not be used since the other pairwise comparison provided by w_1 (of $2 < 3$) is incompatible with the perception of w_3 (of $3 < 2$). Therefore, given the evidence of w_3 disagreeing with w_1 on the pairwise comparisons in $Q(w_1) \cap Q(w_3)$ [8], it is more likely that w_3 will disagree with w_1 for other comparisons as well.

- Suppose we had a comparison including node '1' given by some worker w_j that was not incompatible with worker w_3's perception. For instance, in the example in Fig. 3, let there be a fourth worker, w_4, who was queried the subset Q_1 (i.e. subset $\{1, 2, 3\}$). Since w_4 is compatible with w_3, therefore w_4's judgement on nodes '2' and '3' must be $3 < 2$. However, let w_4's other judgements over Q_1 be $(1 < 2)$ and $(3 < 1)$. As a result, the alternate (and compatible) orderings we have available are:

$$w_3 \rightarrow 3 < 4 < 2$$
$$w_4 \rightarrow 3 < 1 < 2$$

  As is evident, it is still impossible to infer a global ordering from these two compatible orderings, whose union covers the entire set $\mathcal{S}$, since we have no notion of the relative ordering within the nodes '1' and '4'.

One possibility of circumventing this issue is by an adaptive sampling technique that ensures that all pairwise comparisons required for any particular global ordering are available by workers whose perceptions are compatible with that ordering. Trying to infer multiple global sort orders *post-hoc*, however, is difficult since we are bound to be left with lots of incomplete orderings.

We could still employ some sort of a filtering mechanism for any incomplete orderings, or we could output incomplete orderings by themselves, along with global orders, and we could perhaps discuss the various merits and demerits of using these techniques in the forthcoming meeting.

3.3 Issue with the Previous Algorithm

For completeness, we quote the algorithm proposed by Ayush earlier, before pointing out some issues with the correctness of the algorithm.

1. Initialise all comparisons provided by workers as unexplained.
2. Select the sort order that explains the maximum number of unexplained comparisons
3. Mark all comparisons explained by this sort order as explained

Because we try and remove the minimum number of edges from the directed graph of worker responses, we essentially remove individual pairwise comparisons between provided by workers, out of all the pairwise judgements that that worker may have provided. The issue with this is that if on some pairwise comparisons, two workers' perceptions differ, then it is flawed in principle to be combining other judgements of

[8]The notation $Q(w_k)$ refers to the query subset that was answered by worker w_k.

theirs on other pairwise comparisons that were not attempted by both.

This arguments is similar to the one provided earlier, for the example in Figure 3, about why it is inappropriate to combine w_1's response and w_3's response, since they are incompatible on the pairwise comparisons in $Q(w_1) \cap Q(w_3)$.

References

[1] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972.