

# Multiple Intents Re-Ranking\*

Yossi Azar<sup>†</sup>

Iftah Gamzu<sup>‡</sup>

Xiaoxin Yin<sup>§</sup>

## Abstract

One of the most fundamental problems in web search is how to re-rank result web pages based on user logs. Most traditional models for re-ranking assume each query has a single intent. That is, they assume all users formulating the same query have similar preferences over the result web pages. It is clear that this is not true for a large portion of queries as different users may have different preferences over the result web pages. Accordingly, a more accurate model should assume that queries have multiple intents.

In this paper, we introduce the *multiple intents re-ranking* problem. This problem captures scenarios in which some user makes a query, and there is no information about its real search intent. In such cases, one would like to re-rank the search results in a way that minimizes the efforts of all users in finding their relevant web pages. More formally, the setting of this problem consists of various types of users, each of which interested in some subset of the search results. Moreover, each user type has a non-negative profile vector. Consider some ordering of the search results. This order determines a position for each search result, and induces a position vector of the results relevant to each user type. The overhead of a user type is the dot product of its profile vector and its induced position vector. The goal is to order the search results as to minimize the average overhead of the users.

Our main result is an  $O(\log r)$ -approximation algorithm for the problem, where  $r$  is the maximum number of search results that are relevant to any user type. The algorithm is based on a new technique, which we call harmonic interpolation. In addition, we consider two important special cases. The first case is when the profile vector of each user type is non-increasing. This case is a generalization of the well-known *min-sum set cover* problem. We extend the techniques of Feige, Lovász and Tetali (Algorithmica '04), and present an algorithm achieving 4-approximation. The second case is when the profile vector of each user type is non-decreasing. This case generalizes the *minimum latency set cover* problem, introduced by Hassin and Levin (ESA '05). We devise an LP-based algorithm that attains 2-approximation.

---

\*An extended abstract of this paper appeared in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 669–678, 2009.

<sup>†</sup>Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA and Blavatnik School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Supported by the Israel Science Foundation. Email: [azar@tau.ac.il](mailto:azar@tau.ac.il).

<sup>‡</sup>Blavatnik School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Supported by the Binational Science Foundation, by the Israel Science Foundation, by the European Commission under the Integrated Project QAP funded by the IST directorate as Contract Number 015848, and by a European Research Council (ERC) Starting Grant. This work was done when the author was visiting Microsoft Research. Email: [iftgam@tau.ac.il](mailto:iftgam@tau.ac.il).

<sup>§</sup>Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA. Email: [xyin@microsoft.com](mailto:xyin@microsoft.com).

## 1 Introduction

**Web search and ranking.** Web search has become an important part of people’s lives. There are many mature approaches for ranking result web pages for a query that are based on the page contents and the hyperlink structure of the web. Nevertheless, the users are the final judges of the search results, and thus, a search engine must utilize their behavior to better understand the intents of queries and the relevance of result web pages. In consequence, one of the most fundamental problems in web search is how to re-rank result web pages based on user logs.

A user usually expresses his judgments on the relevance of result web pages by clicking on certain results. There have been studies on how to infer user preferences and re-ranking result web pages based on user clicks [1, 6]. In these studies it is assumed that different users have the same preference on the result web pages, that is, they have the same intent for a query. However, this is not true for a large portion (if not the majority) of queries. For example, there are many ambiguous queries, such as ‘Giant’, which may refer to a bicycle company, a food store chain, or even a movie. According to [17], about 16% of the queries are ambiguous. Even for a common non-ambiguous query such as ‘Hong Kong’, some users may be searching for attractions in Hong Kong, some for maps of Hong Kong, and others for how to travel to Hong Kong.

For a query with multiple intents, a good ranking system should provide result web pages that cover each major intent. Furthermore, it seems that minimizing the efforts of all users in finding the web pages that satisfy their needs is of the essence. When performing web search, a user usually reads the result items from top to bottom [11], and clicks on the results that are relevant to his search intent. In most situations, reading the result items helps the user to decide the relevance of each result web page. Correspondingly, the time a user spends on reading the result items is the overhead in web search, which should be minimized.

**A theoretical model.** We propose an approach for minimizing the efforts of all users in finding the web pages of their interest. We model the problem as follows. There is a set of result items for a certain query, and there are several user types (of some known proportion), each of which is interested in a known subset of the items. The general question is in which order to present the result items as to minimize the average overhead of the users. The overhead of a user type depends on the positions of its relevant items in the ordering, and its characterizing profile vector. A profile vector is a non-negative weight vector whose length equals the number of relevant items of the underlying user type. Consider some user type. Any ordering of all result items induces an increasingly sorted position vector on the items relevant to that user type. For instance, the first entry in this vector corresponds to the time it takes the user type to find its first relevant item according to the ordering. The overhead of a user type is defined as the dot product of its profile vector and its induced position vector.

As examples of the suggested model, we consider two extreme settings. The first setting considers the overhead of all user types to be proportional to the time it takes them to find their first relevant item in the ordering. The profile vector of each user type has a  $\langle 1, 0, \dots, 0 \rangle$ -structure. Such profile usually applies to user types whose query is navigational, that is, a query looking for one relevant web page [10]. The objective in this case is to find an ordering that minimizes the (weighted) average position of the first relevant item of the users. Notably, this special setting is equivalent to the *min-sum set cover* problem, studied by Feige, Lovász and Tetali [7]. The second setting regards the overhead of all user types to be proportional to the time it takes them to find their last relevant item. The profile vector of each user type has a  $\langle 0, \dots, 0, 1 \rangle$ -structure. Such profile commonly relates to user types whose query is informational, namely, a query aiming to obtain data or information from a set of relevant web page [10]. The goal in this case is to find an ordering that minimizes the (weighted) average position of the last relevant item of the user types. This special setting captures the *minimum latency set cover* problem, considered by Hassin and Levin [9].

Formally, we model the *multiple intents re-ranking* problem as follows. The input of this problem consists of a hypergraph  $H(V, E)$  such that  $n = |V|$ , and every hyperedge  $e \in E$  has an associated non-negative weight vector  $w(e) = \langle w_1(e), \dots, w_{r(e)}(e) \rangle$ . Note that  $r(e)$  denotes the rank of  $e$ , namely, the number of vertices it contains. Essentially, each vertex of the hypergraph models a result item, every hyperedge corresponds to a user type, and the non-negative weight vector of a hyperedge is the profile vector of that user type. The objective is to find a linear ordering of the vertices, that is, a permutation  $\varphi : V \rightarrow [n]$  that minimizes the weighted cover time of the hyperedges. Specifically, denoting by  $c_i^\varphi(e) = \min_j \{ |\{\varphi^{-1}(1), \dots, \varphi^{-1}(j)\} \cap e| = i \}$  the linear order index of the vertex that is the  $i$ -th to cover  $e$  according to  $\varphi$ , the goal is to find an ordering  $\varphi$  that minimizes  $\sum_{e \in E} \sum_{i=1}^{r(e)} w_i(e) c_i^\varphi(e)$ . It is worthy to note that the sum of entries of a weight vector may be interpreted as the proportion of the matching user type in the population.

## 1.1 Our results

Our findings can be briefly summarized as follows:

- Our main result is an  $O(\log r)$ -approximation algorithm for the problem in its utmost generality, that is, when the weight vector of every hyperedge has no particular structure. Note that  $r$  denotes the maximal rank of a hyperedge, which is typically much smaller than  $n$ . The algorithm is based on a new technique, which we call harmonic interpolation. To the best of our knowledge, there are no previous results in this setting.
- We present a 4-approximation weight reduction greedy algorithm for the important special case in which the weight vector of every hyperedge is monotonically non-increasing. This case generalizes min-sum set cover, and our result extends the techniques of Feige, Lovász and Tetali [7]. Note that our algorithm is greedy with respect to the gradient of the weight vectors and not their total weight. In particular, the latter algorithm fails to provide fair approximation. Also note that for the special case of min-sum set cover both algorithms are essentially identical.
- We provide a 2-approximation algorithm based on a deterministic LP rounding for the other important special case in which the weight vector of every hyperedge is monotonically non-decreasing. This case generalizes minimum latency set cover, and in particular, our result improves the  $\epsilon$ -approximation algorithm to this problem due to Hassin and Levin [9].<sup>1</sup>

It is worthy to note that all the algorithms are different, and require different analysis methods.

## 1.2 Technical highlights

Our primary technical highlight is the harmonic interpolation technique, which is utilized in the arbitrary weight vectors case. It is motivated by the observation that the weight reduction greedy algorithm, which achieves 4-approximation for non-increasing weight vectors, miserably fails to provide suitable approximation for the general case. In particular, a greedy type algorithm fails to differentiate between different indicator vectors (i.e., weight vectors consisting of exactly one non-zero entry) until it considers a vertex that corresponds to a non-zero entry. We resolve this problem by introducing the harmonic interpolation technique, which spreads the weights of each vector in a non-uniform way. We believe that this method could be useful in solving more problems, similarly to the well-established and nutritiously-used exponential weights method (see, e.g., [16, 8]). It is worthy to mention that for the special case of non-decreasing weight vectors, we use a completely different technique, which is based on LP relaxation, and deterministic rounding

---

<sup>1</sup>In a later version of their paper, Hassin and Levin indicate that their result can be improved using a known 2-approximation algorithm for the minimizing weighted sum of job completion times on a single machine.

procedure. This result has some similarities to the LP formulation of Queyranne [14], and the rounding heuristic of Schulz [15].

### 1.3 Related work

The min-sum set cover is a special case of multiple intents re-ranking in which the weight vectors of all hyperedges have a  $\langle 1, 0, \dots, 0 \rangle$ -structure. Feige, Lovász and Tetali [7] developed a greedy algorithm that approximates it to within a ratio of 4. Essentially, this result was implicit in the work of Bar-Noy et al. [3], but was significantly simplified. Feige, Lovász and Tetali also proved that it is NP-hard to approximate it to within a ratio of  $4 - \epsilon$ , for every  $\epsilon > 0$ . Prior to this result, Bar-Noy, Halldórsson and Kortsarz [4] established the tightness of the greedy algorithm. The min-sum set cover problem have been generalized in different ways, and found many interesting applications in various fields such as peer to peer networks [5], data streams and database query processing optimization [2, 13], and even in learning theory [12].

The minimum latency set cover is a special case of multiple intents re-ranking in which the weight vectors of all hyperedges has a  $\langle 0, \dots, 0, 1 \rangle$ -structure. Hassin and Levin [9] introduced this problem, proved that it is NP-Hard in a strong sense, and devised an  $\epsilon$ -approximation algorithm.

## 2 Arbitrary Weight Vectors

In this section, we study the multiple intents re-ranking problem in its utmost generality, that is, when the weight vector of every hyperedge has no particular structure. We develop an  $O(\log r)$ -approximation algorithm, where  $r$  is the maximal rank of a hyperedge.

### 2.1 The algorithm

Prior to describing the finer details of our algorithm, we make an effort to motivate it. For this purpose, it would be convenient to view the linear ordering as discretized time, and evaluate the contribution of hyperedge  $e$  to the objective function in each time step. One can verify that the objective function is incremented by  $w_i(e)$  in every time step until  $e$  is covered for the  $i$ -th time. Particularly, if  $e$  is covered by  $i - 1$  vertices at the beginning of time step  $t$  then its contribution to the objective function at that step is  $\sum_{j=i}^{r(e)} w_j(e)$ . This gives rise to a weight reduction greedy algorithm which selects, in each time step, a vertex that minimizes the overall weight contribution of the hyperedges for the subsequent step. Unfortunately, simple examples demonstrate that such algorithm fails to guarantee fair approximation (see, e.g., Appendix A.1). These examples target the limited view of the algorithm, which only considers the current maximal weight reduction, and does not take into account future weight reduction potentials. We overcome this difficulty by approximating each weight vector using a harmonic interpolation, and only then employ the above-mentioned greedy strategy. Intuitively, the harmonic interpolation enables the algorithm to leverage knowledge about future weight reduction potentials of the hyperedges.

Algorithm Harmonic Ranking, formally described below, consists of two logical blocks. The first is a weight modification block, in which a harmonic interpolation vector  $\tilde{w}(e)$  that approximates  $w(e)$  is produced, for every hyperedge  $e$ . The second block is built from a sequence of greedy steps that establish the linear ordering of the vertices. Specifically, the algorithm maintains one variable for each hyperedge, and one variable for each vertex. The variable corresponding to hyperedge  $e$  represents the current weight reduction potential of  $e$  (and not the total remaining weight of  $e$ ). More precisely, the weight reduction potential of  $e$ , which is already covered by  $i - 1$  vertices, is  $\tilde{w}_i(e)$ , for every  $i \in [r(e)]$ , and 0 if  $e$  is completely covered, i.e., all the vertices incident on  $e$  were selected in previous steps. Analogously, the variable corresponding to vertex  $v$  designates the current weight reduction potential of  $v$ , which is equal to the overall weight reduction potential of the hyperedges incident on  $v$ . In each step, the algorithm greedily chooses

a non-selected vertex whose weight reduction potential is maximal, and affixes it to the linear ordering. Note that using a greedy selection rule with respect to total remaining weight as the second block of the algorithm fails to provide fair approximation.

---

**Algorithm 1** Harmonic Ranking

---

▷ *Phase I: harmonic interpolation.*

- 1: **for all**  $e \in E$  and  $i \in [r(e)]$  **do**
- 2:     Let  $\tilde{w}_i(e) = \sum_{j=i}^{r(e)} \frac{w_j(e)}{j-i+1}$
- 3: **end for**

▷ *Phase II: greedy weight reduction.*

- 4:  $S \leftarrow \emptyset$
- 5: **for**  $t \leftarrow 1$  to  $n$  **do**
- 6:     **for all**  $e \in E$  **do**
- 7:          $i \leftarrow |\{\varphi^{-1}(1), \dots, \varphi^{-1}(t-1)\} \cap e| + 1$
- 8:         **if**  $i \in [r(e)]$  **then**  $\rho(e) \leftarrow \tilde{w}_i(e)$  **else**  $\rho(e) \leftarrow 0$  **end if**
- 9:     **end for**
- 10:      $\rho(v) \leftarrow \sum_{e|v \in e} \rho(e)$
- 11:     Let  $v' \in V \setminus S$  be a vertex having maximal  $\rho(v)$
- 12:      $S \leftarrow S \cup \{v'\}$
- 13:      $\varphi(v') \leftarrow t$
- 14: **end for**

---

## 2.2 Analysis

In what follows, we analyze the approximation guarantee of the algorithm. For ease of presentation, we concentrate on instances of the problem in which the weight vector of every hyperedge is an indicator vector, namely, a vector consisting of exactly one entry different than 0. One can verify that this setting is equivalent to the arbitrary weights case. The following reduction may be used to demonstrate this equivalence: given an arbitrary weights instance, replace every hyperedge having a weight vector of  $\langle w_1, \dots, w_r \rangle$  with  $r$  hyperedges on the same set of vertices such that one of them has a weight vector of  $\langle w_1, 0, \dots, 0 \rangle$ , one has a weight vector of  $\langle 0, w_2, 0, \dots, 0 \rangle$ , and so on up to the last hyperedge, which has a weight vector of  $\langle 0, \dots, 0, w_r \rangle$ .

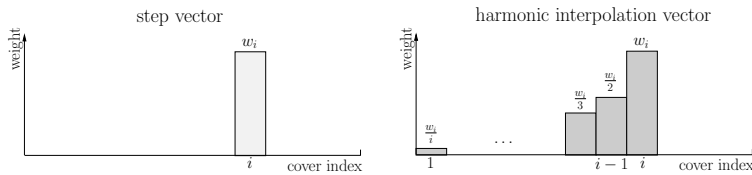


Figure 1: The term *harmonic interpolation* is inspired by the observation that if the weight vector of hyperedge  $e$  is  $w(e) = \langle 0, \dots, 0, w_i, 0, \dots, 0 \rangle$  then  $\tilde{w}(e) = \langle w_i/i, \dots, w_i/2, w_i, 0, \dots, 0 \rangle$ .

We begin by introducing the following notation:

- Let  $\text{OPT}$  and  $\text{ALG}$  be the weight of the cover induced by the optimal linear ordering and the linear ordering constructed by the algorithm with respect to the original weight vectors.
- Let  $(e, i)$  be a *potential pair* consisting of a hyperedge  $e$  and a cover index  $i$ . Intuitively, such pair stands for the possibility that  $e$  was covered for the  $i$ -th time. Let  $E_t$  be the collection of potential pairs covered during step  $t$  of the algorithm. Specifically, each  $(e, i) \in E_t$  stands

for a hyperedge  $e$  that is covered for its  $i$ -th time during step  $t$ . Similarly, let  $R_t$  be the collection of potential pairs not covered prior to step  $t$  of the algorithm. More formally,  $R_t = R \setminus \bigcup_{j=1}^{t-1} E_j$ , where  $R = \{(e, i) : e \in E \text{ and } i \in [r(e)]\}$  is the set of all valid pairs.

- Let  $P_t = \sum_{(e,i) \in R_t} w_i(e) / \sum_{(e,i) \in E_t} \tilde{w}_i(e)$  be the *penalty* of step  $t$  of the algorithm.

Note that we can reinterpret the solution value of the algorithm using the suggested notation. Particularly, one can validate that

$$\text{ALG} = \sum_{t=1}^n \sum_{(e,i) \in E_t} t \cdot w_i(e) = \sum_{t=1}^n \sum_{(e,i) \in R_t} w_i(e) = \sum_{t=1}^n \sum_{(e,i) \in E_t} P_t \cdot \tilde{w}_i(e).$$

The first term holds by the definition of the objective function. The second term follows by recalling that the objective function is incremented by  $w_i(e)$  in every time step until  $e$  is covered for the  $i$ -th time. Finally, the last term results by noticing that  $\sum_{(e,i) \in E_t} P_t \cdot \tilde{w}_i(e) = \sum_{(e,i) \in R_t} w_i(e)$ .

**Theorem 2.1.** *Algorithm Harmonic Ranking constructs a linear ordering whose induced weight is no more than  $O(\log r)$  times the optimal one, where  $r$  is the maximal rank of a hyperedge.*

**Proof.** We begin by introducing two histograms corresponding to the optimal solution and the solution of the algorithm:

**The optimal solution histogram.** This histogram is related to the optimal solution with respect to the original weight vectors. It consists of  $|R|$  columns, one for each potential pair. The columns are ordered according to the steps in which the corresponding pairs were covered by the optimal algorithm. Furthermore, the width of each column is the original weight of the corresponding pair, and the height of each column is the step in which that pair was covered. Notice that the histogram is non-decreasing by definition, and that the area beneath it is exactly OPT. Figure 2(a) provides an illustration of this histogram.

**The algorithm's solution histogram.** This histogram corresponds to the solution of the algorithm with respect to the original weight vectors. Similarly to the previous histogram, this histogram also consists of  $|R|$  columns, one for each potential pair. Additionally, the columns are ordered according to the steps in which the corresponding pairs were covered by the algorithm. However, the width of each column is the harmonic weight of the corresponding pair, and the height of each column is the penalty of the step in which that pair was covered. In particular, this implies that this histogram is not necessarily monotonic. Take notice that the area beneath the histogram is precisely ALG. Figure 2(b) presents an illustration of this histogram.

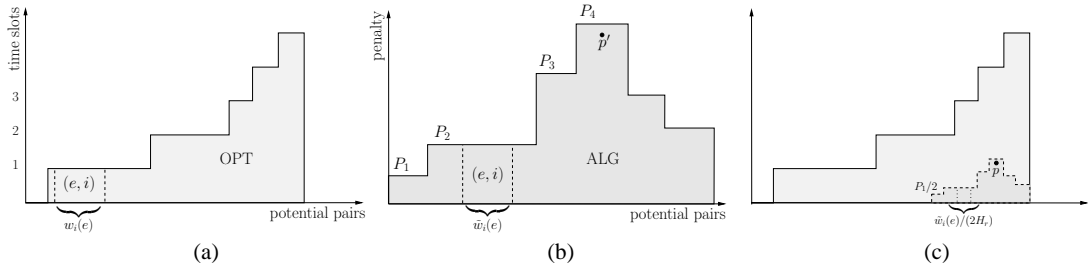


Figure 2: (a) The optimal solution histogram; (b) The algorithm's solution histogram; (c) The optimal solution histogram vs. the shrunken algorithm's solution histogram.

Note that the overall width of the second histogram is at least as large as the width of the first histogram since it is defined with respect to the harmonic weights and not the original weights. Nevertheless, one can easily verify that it is larger by no more than a multiplicative factor of

$H_r$ , where  $H_r = \sum_{j=1}^r 1/j$  is the  $r$ -th harmonic number. We next argue that the area beneath the second histogram is at most  $4H_r$  times larger than the area beneath the first histogram. For this purpose, let us consider the transformation that shrinks the width and height of each column of the second histogram by a factor of  $2H_r$  and 2, respectively. Specifically, the column corresponding to a potential pair  $(e, i)$  that was covered during step  $t$  of the algorithm has a width of  $\tilde{w}_i(e)/(2H_r)$ , and a height of  $P_t/2$ . We also align this histogram to the right lower boundary of the first histogram. Figure 2(c) illustrates the transformation and the alignment. We claim that this shrunken histogram is completely contained within the first histogram. Thus, its area is no more than the area beneath the first histogram, implying that  $\text{ALG}/(4H_r) \leq \text{OPT}$ .

Consider an arbitrary point  $p'$  in the second histogram, and assume without loss of generality that it lies in the column corresponding to a potential pair  $(e', i')$  covered during step  $t'$  of the algorithm. Correspondingly, the height of  $p'$  is at most  $P_{t'}$ , and its distance from the right side boundary is at most  $\Delta_{t'} = \sum_{(e,i) \in R_{t'}} \tilde{w}_i(e)$ . We focus on the point  $p$ , which is the mapping of  $p'$  in the shrunken histogram, and prove that it lies within the first histogram. Notice that the height of  $p$  is at most  $P_{t'}/2$ , and its distance from the right boundary of the first histogram is at most  $\Delta_{t'}/(2H_r)$ . Accordingly, it is sufficient that we demonstrate that by time step  $\lfloor P_{t'}/2 \rfloor$ , the optimal algorithm has at least  $\Delta_{t'}/(2H_r)$  weight to cover. For this purpose, let us consider the following truncation of the original weights instance. In the truncated instance, all the vertices selected by the algorithm up to step  $t'$  are given for free, and the indicator vector of any hyperedge  $e$  is spread. Specifically, suppose  $w(e) = \langle 0, \dots, 0, w_i, 0, \dots, 0 \rangle$  and the algorithm covered  $j-1$  vertices of  $e$  until the beginning of step  $t'$ . Notice that the harmonic interpolation guarantees that the next covering weight of the algorithm for  $e$  is  $w' = w_i/(i-j+1)$ . The spreading process replaces  $w(e)$  with a spread vector  $w'(e) = \langle 0, \dots, 0, w', \dots, w', 0, \dots, 0 \rangle$ , where the non-zero weights appear in the entries  $j$  to  $i$ . Figure 3 illustrates the spreading process.

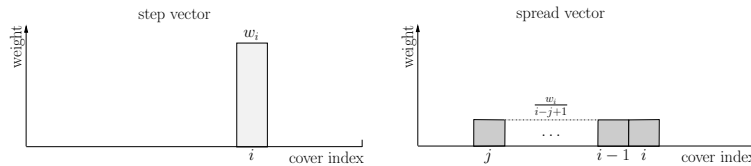


Figure 3: The spreading process. Notice that the process does not change the overall weight of an edge, but only changes the partition of the weight to its covering indices.

The crucial observation one should make regarding this truncated instance is that the linear ordering constructed by the optimal algorithm with respect to the original weights induces a solution for this instance whose matching histogram is completely contained within the optimal solution histogram when aligned to its lower right boundary. Hence, by proving that any algorithm applied to the truncated instance has at least  $\Delta_{t'}/(2H_r)$  weight to cover by time step  $\lfloor P_{t'}/2 \rfloor$ , one obtains that the same bound follows for the optimal algorithm applied to the original instance. Note that the above-mentioned observation results from the following lemma, whose proof appears in Appendix A.2, by noticing that the modifications used to generate the truncated instance can be interpreted as a sequence of the basic transformations described in the lemma. In particular, one should notice that the effect of giving a vertex for free can be interpreted as decreasing the weight of all potential pairs covered by this vertex to 0.

**Lemma 2.2.** *Consider a monotonically non-decreasing histogram having the same interpretation of the axes as the optimal solution histogram, and suppose we modify the weight vector  $w(e)$  of some hyperedge  $e$  by applying one of the following transformations:*

- Decrease the weight of  $w_i(e)$ .
- Transfer part of the weight of  $w_i(e)$  to  $w_j(e)$ , where  $j \leq i$ .

The histogram resulting from the application of the transformation is monotonically non-decreasing, and completely contained within the primary histogram when aligned to its lower right boundary.

Let us focus on the set of potential pairs  $R_{t'}$  in the truncated instance. Notice that any non-selected vertex chosen by any algorithm in any step cannot reduce the weight contributed to the objective function by more than  $\sum_{(e,i) \in E_{t'}} \tilde{w}_i(e)$ . This follows from the greedy selection rule, which ensures that the vertex chosen maximizes the weight reduction, and from the construction of the spread vectors, which guarantees that as the algorithm progresses its weight reduction from covering the same edge cannot increase. Consequently, until time step  $\lfloor P_{t'}/2 \rfloor$ , any algorithm may cover no more than  $\lfloor P_{t'}/2 \rfloor \cdot \sum_{(e,i) \in E_{t'}} \tilde{w}_i(e) \leq \sum_{(e,i) \in R_{t'}} w_i(e)/2$  weight. The construction of the truncated instance guarantees that the set of potential pairs  $R_{t'}$  has an overall weight of exactly  $\sum_{(e,i) \in R_{t'}} w_i(e)$ . This implies that at least  $\sum_{(e,i) \in R_{t'}} w_i(e)/2 \geq \Delta_{t'}/(2H_r)$  of its weight is left uncovered, where the inequality follows since the harmonic interpolation increases the weight of every hyperedge by no more than a multiplicative factor of  $H_r$ . ■

### 3 Non-Increasing Weight Vectors

In this section, we consider the special case of the multiple intents re-ranking problem when the weight vector of every hyperedge is monotonically non-increasing, namely,  $w_1(e) \geq \dots \geq w_{r(e)}(e)$ , for every  $e \in E$ . We prove that a weight reduction greedy algorithm, based on the second phase of algorithm **Harmonic Ranking** (i.e., lines 4-14) presented in Section 2, attains constant approximation ratio of 4. Note that the weight reduction greedy algorithm is identical to the aforesaid greedy phase with the exception that it is applied to the original weight vectors instead of the harmonic vectors. It seems worthy to recall that the approximation guarantee of this weight reduction greedy algorithm was unbounded in the arbitrary weights case (see, e.g., Appendix A.1).

#### 3.1 Analysis

In the following, we analyze the performance guarantee of the greedy algorithm. The analysis follows a similar line of argumentation as the analysis of algorithm **Harmonic Ranking**. Still, it consists of several key differences which result from the non-increasing nature of the weight vectors, and the omission of the harmonic interpolation phase. We begin by slightly refining the notation introduced in Subsection 2.2. Basically, we utilize the same notation, but naturally adjust every term which used harmonic weights to use the original weights. For instance, the penalty of step  $t$  of the algorithm is now defined as  $P_t = \sum_{(e,i) \in R_t} w_i(e) / \sum_{(e,i) \in E_t} w_i(e)$ , and consequently, the solution value of the algorithm can also be reinterpreted as  $\text{ALG} = \sum_{t=1}^n \sum_{(e,i) \in E_t} P_t \cdot w_i(e)$ .

**Theorem 3.1.** *The weight reduction greedy algorithm constructs a linear ordering whose induced weight is no more than 4 times the optimal one.*

**Proof.** We consider the same histograms defined in the proof of Theorem 2.1. The first is a histogram corresponding the optimal solution, and the other is a histogram corresponding to the solution of the greedy algorithm. Note that the width of each column of the latter histogram is the weight of the corresponding pair, and not the harmonic weight as in the mentioned proof.

We argue that the area beneath the second histogram is at most 4 times larger than the area beneath the first histogram. For this purpose, we first shrink the width and height of each column of the second histogram by half, and then align the resulting histogram to the right lower boundary of the first histogram. Note that the column corresponding to a potential pair  $(e, i)$  that was covered during step  $t$  of the algorithm has a width of  $w_i(e)/2$  and a height of  $P_t/2$  in the shrunken histogram. We now claim that this histogram is completely contained within the first histogram, implying that  $\text{ALG}/4 \leq \text{OPT}$ , which proves the theorem.



oracle. This oracle can be used in conjunction with the ellipsoid algorithm to solve the linear program in polynomial-time. Given a fractional solution  $(x, y)$  to the linear relaxation, the separation oracle tests the constraints as follows:

- Constraint (1): For every hyperedge  $e = (u_1, \dots, u_{r(e)}) \in E$ , the oracle initially sorts the  $x$ -variables corresponding to the vertices of  $e$ . Let us assume without loss of generality that  $x_{u_1} \leq \dots \leq x_{u_{r(e)}}$ . Then, it verifies that  $y_e \geq \sum_{i=1}^{r(e)} w_i(e)x_{u_i}$ .
- Constraint (2): The oracle initially sorts all the  $x$ -variables. Let us assume without loss of generality that  $x_{v_1} \leq \dots \leq x_{v_n}$ . Subsequently, it validates that  $\sum_{i=1}^k x_{v_i} \geq (k^2 + k)/2$ , for every  $k \in [n]$ .

**The deterministic rounding procedure.** Consider an optimal fractional solution  $(x^*, y^*)$  to the relaxation of (IP), and assume without loss of generality that  $x_{v_1}^* \leq \dots \leq x_{v_n}^*$ . We define a linear ordering of the vertices according to sorted order induced by  $x^*$ . More precisely, we set  $\varphi(v_i) = i$ , for every  $i \in [n]$ .

## 4.2 Analysis

In the following, we analyze the properties of the algorithm. We begin by arguing that the integer program describes our setting accurately. Essentially, the following lemma claims that an optimal solution to the problem under consideration designates a feasible solution of the integer program.

**Lemma 4.1.** *An optimal solution for the multiple intents re-ranking problem with monotonically non-decreasing weight vectors defines a feasible solution for (IP).*

**Proof.** Consider an optimal linear ordering  $\varphi^*$  for the problem under consideration, and define a solution  $(x, y)$  for (IP) as follows:  $x_v = \varphi^*(v)$ , for every  $v \in V$ , and  $y_e = \sum_{i=1}^{r(e)} w_i(e)c_i^{\varphi^*}(e)$ , for every  $e \in E$ . Recollect that  $c_i^{\varphi^*}(e)$  is the index of the vertex that is the  $i$ -th to cover  $e$  according to  $\varphi^*$ . In the following, we prove that each of the constraints are satisfied by this solution.

Consider a constraint of type (1) whose underlying hyperedge is  $e = (u_1, \dots, u_{r(e)}) \in E$ . Assume without loss of generality that the ordering induced by  $\varphi^*$  on the vertices of  $e$  satisfies  $\varphi^*(u_1) \leq \dots \leq \varphi^*(u_{r(e)})$ , and let  $\varphi^*(e) = \langle x_{u_1}, \dots, x_{u_{r(e)}} \rangle = \langle \varphi^*(u_1), \dots, \varphi^*(u_{r(e)}) \rangle$ . Recall that  $w(e)$  is monotonically non-decreasing. This implies, in conjunction with the monotonically non-decreasing nature of  $\varphi^*(e)$ , that  $w(e) \cdot \varphi^*(e)$  is maximal with respect to the dot product of any permutation of  $w(e)$  and  $\varphi^*(e)$ . A formal proof of the last statement appears in Appendix B.1. Consequently, we get that

$$y_e = \sum_{i=1}^{r(e)} w_i(e)c_i^{\varphi^*}(e) = \sum_{i=1}^{r(e)} w_i(e)x_{u_i} \geq \sum_{i=1}^{r(e)} w_i(e)x_{\pi(u_i)},$$

for every  $\pi \in \Pi_{r(e)}$ . Accordingly, any constraint of type (1) is satisfied by the solution  $(x, y)$ .

Now, consider a constraint of type (2), and let  $S \subseteq V$  be the subset that determines this constraint. It is clear, by the linearity of the ordering, that  $\sum_{v \in S} x_v \geq 1 + \dots + S = (|S|^2 + |S|)/2$ . Thus, any constraint of type (2) is satisfied by the solution  $(x, y)$ . ■

We proceed by establishing the correctness of the separation oracle. One can easily verify that the oracle runs in polynomial-time. Therefore, we are left to prove that the oracle identifies a violated constraint in case of an infeasible solution.

**Lemma 4.2.** *Given an infeasible solution  $(x, y)$  to the linear relaxation of (IP), the separation oracle identifies some violated constraint.*

**Proof.** We consider two cases, depending on the type of constraints violated by the given solution. Note that if the solution violates constraints of both types then any case is applicable.

Suppose a constraint of type (1) is violated by the given solution, and let  $e \in E$  and  $\pi \in \Pi_{r(e)}$  be a hyperedge and a permutation that designate the violated constraint. Let us consider the test of the oracle for hyperedge  $e$ . Essentially, the oracle checks the underlying constraint with respect to one specific permutation  $\pi'$ , which is the sorting permutation of the  $x$ -variables corresponding to the vertices of  $e$ . We now argue that  $\sum_{i=1}^{r(e)} w_i(e)x_{\pi'(u_i)} \geq \sum_{i=1}^{r(e)} w_i(e)x_{\pi(u_i)}$ , for any permutation  $\pi$ . Notice that this suggests that if the underlying constraint is violated with respect to  $\pi$  then it must also be violated with respect to  $\pi'$ , and thus, the oracle identified some violated constraint. The above-mentioned argument follows by recalling that both the weight vector of  $e$  and the permutation vector determined by  $\pi'$  are monotonically non-decreasing, and thus, their dot product is maximal. A formal proof of the fact that the dot product of two non-negative vectors is maximal if their ordering is consistent, as well as a formal definition of consistent ordering, appears in Appendix B.1.

Now, suppose a constraint of type (2) is violated by the given solution, and let  $S \subseteq V$  be a subset of size  $k$  that determines the violated constraint. Let us consider the test of the oracle for subsets of size  $k$ . Essentially, the oracle checks the underlying constraint with respect to the set  $L = \{x_{v_1}, \dots, x_{v_k}\}$  of  $k$  variables having the lowest values. It is clear that  $\sum_{v \in L} x_v \leq \sum_{v \in S} x_v$ . Hence, if the underlying constraint is violated with respect to  $S$  then it must also be violated with respect to  $L$ , and therefore, the oracle identified some violated constraint. ■

We turn to prove that the algorithm attains approximation ratio of  $2 - 2/(n+1)$ . The next lemma sets an upper bound on the linear order index of each vertex in terms of the corresponding fractional variable.

**Lemma 4.3.**  $\varphi(v_k) \leq \left(2 - \frac{2}{k+1}\right)x_{v_k}^*$ , for every  $k \in [n]$ .

**Proof.** Let us consider the set of variables  $S = \{x_{v_1}^*, \dots, x_{v_k}^*\}$ . Constraint (2) guarantee that  $\sum_{v \in S} x_v^* \geq (k^2 + k)/2$ . Furthermore, recall that  $x_{v_k}^* \geq \dots \geq x_{v_1}^*$ , and hence,  $x_{v_k}^* \geq (k+1)/2$ . This implies that  $\varphi(v_k) = k \leq (2 - 2/(k+1))x_{v_k}^*$ , where the equality holds by the properties of the rounding procedure. ■

**Theorem 4.4.** *The algorithm constructs a linear ordering whose induced weight is no more than  $2 - \frac{2}{n+1}$  times the optimal one.*

**Proof.** Consider some hyperedge  $e = (u_1, \dots, u_{r(e)})$ , and let us assume without loss of generality that the ordering induced by  $\varphi$  on the vertices of  $e$  satisfies  $\varphi(u_1) \leq \dots \leq \varphi(u_{r(e)})$ . One can see that  $y_e^\varphi$ , the weighted cover time of  $e$  under the linear ordering  $\varphi$ , satisfies

$$y_e^\varphi = \sum_{i=1}^{r(e)} w_i(e)\varphi(u_i) \leq \left(2 - \frac{2}{n+1}\right) \sum_{i=1}^{r(e)} w_i(e)x_{u_i}^* \leq \left(2 - \frac{2}{n+1}\right) y_e^*,$$

where the first inequality follows from Lemma 4.3, and the last one results from constraint (1) instantiated with the identity permutation  $\pi$ , which sets  $\pi(u_i) = u_i$ , for every  $u_i \in E$ . Now, the theorem follows by recalling that the objective is to minimize the total weighted cover time of all the hyperedges. ■

We conclude this subsection by arguing that our analysis is tight. Namely, we demonstrate that the integrality gap of the integer program is  $2 - 2/(n+1)$ .

**Lemma 4.5.** *The integrality gap of (IP) is exactly  $2 - \frac{2}{n+1}$ .*

**Proof.** Consider the hypergraph that consists of a single hyperedge  $e$  on  $n$  vertices having weight vector of  $\langle 0, \dots, 0, 1 \rangle$ . One can verify that there is a feasible fractional solution for this instance whose weight is  $(n + 1)/2$ . Particularly, the fractional solution in this case sets  $x_v = (n + 1)/2$ , for every  $v \in V$ , and  $y_e = (n + 1)/2$ . On the other hand, it is clear that the optimal integral solution has a weight of  $n$ . ■

## 5 Additional Remarks

**Decreasing-increasing weight vectors.** It is natural to ask what can be done for instances whose weight vectors are not confined to the aforementioned monotonic structures, but also not arbitrary. In particular, one immediate setting to consider is when part of the hyperedges have non-increasing weight vectors and the remainder of the hyperedges have non-decreasing vectors. It turns out that this setting also admits constant approximation algorithm. The key idea is to partition the given instance into two sub-instances; one sub-instance consists of all hyperedges having non-increasing weight vectors, while the other contains all hyperedges having non-decreasing vectors. The next step is to approximately solve them by employing the previously-mentioned algorithms. Now, given the two resulting linear orderings, one can unify them into a single linear ordering by means of interleaving. Specifically, the unified linear ordering is generated by alternately choosing one vertex according to the first linear ordering and one vertex according to the second linear ordering. It is easy to validate that the linear order index of each vertex is not greater by more than twice its index in each of the underlying linear orderings. This ensures that the unified ordering achieves constant approximation for the original instance. In fact, one can prove that the unified ordering is 12-approximation. Notably, a simple reduction can extend this result even further. Consider some hyperedge  $e$  having a weight vector  $w(e) = \langle w_1(e), \dots, w_{r(e)}(e) \rangle$ . Suppose we replace  $e$  by  $k$  hyperedges  $e_1, \dots, e_k$  on the same set of vertices such that the weight vector of  $e_i$  is  $w^i(e)$ , and  $w(e) = \sum_{i=1}^k w^i(e)$ . One can verify that any linear ordering for this instance has the same weighted cover value as the original instance, and vice versa. This implies that the interleaving technique may be applied to instances in which some edges have more complex weight vectors structure. For example, weight vectors that are initially non-increasing and from some point non-decreasing (see, e.g., Figure 4). Nevertheless, note that this replacement reduction and interleaving combination cannot be utilized to handle arbitrary weights since there is no way to partition an indicator vector (or a weight vector that is initially non-decreasing and from some point non-increasing) into non-negative non-increasing and non-decreasing weight vectors. The full version of the paper will provide a detailed description of these results.

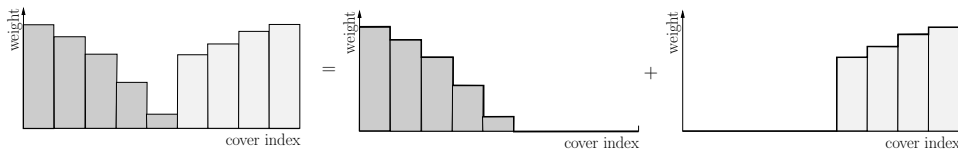


Figure 4: A partition of a hyperedge having a weight vector which is initially non-increasing and then non-decreasing to two monotonic weight vectors.

**Constant weight vectors.** Consider the multiple intents re-ranking problem when the weight vector of every hyperedge is constant. Specifically, the weight vector of each hyperedge  $e$  is  $\langle w_e, \dots, w_e \rangle$ , where  $w_e$  is a non-negative weight characterizing  $e$ . As opposed to the previously mentioned variants of the problem, this variant admits a simple polynomial-time optimal algorithm. Let  $d(v) = \sum_{e|v \in e} w_e$  be the weighted degree of vertex  $v$ . The simple strategy that orders the vertices by non-increasing  $d(v)$  values is optimal. The key observation needed to prove the optimality of the algorithm is that if there is a pair of vertices  $v_1$  and  $v_2$  such that  $v_1$  is ordered

before  $v_2$ , but  $d(v_1) < d(v_2)$  then switching the order of these vertices improves the weighted cover time of the hyperedges. Further details are deferred to the full version of the paper.

## References

- [1] E. Agichtein, E. Brill, and S. T. Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 19–26, 2006.
- [2] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 407–418, 2004.
- [3] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.
- [4] A. Bar-Noy, M. M. Halldórsson, and G. Kortsarz. A matched approximation bound for the sum of a greedy coloring. *Inf. Process. Lett.*, 71(3-4):135–140, 1999.
- [5] E. Cohen, A. Fiat, and H. Kaplan. Efficient sequences of trials. In *Proceedings 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 737–746, 2003.
- [6] G. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 331–338, 2008.
- [7] U. Feige, L. Lovász, and P. Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.
- [8] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings 39th Annual Symposium on Foundations of Computer Science*, pages 300–309, 1998.
- [9] R. Hassin and A. Levin. An approximation algorithm for the minimum latency set cover problem. In *Proceedings 13th Annual European Symposium on Algorithms*, pages 726–733, 2005.
- [10] B. J. Jansen, D. L. Booth, and A. Spink. Determining the informational, navigational, and transactional intent of web queries. *Inf. Process. Manage.*, 44(3):1251–1266, 2008.
- [11] T. Joachims, L. A. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst.*, 25(2), 2007.
- [12] H. Kaplan, E. Kushilevitz, and Y. Mansour. Learning with attribute costs. In *Proceedings 37th Annual ACM Symposium on Theory of Computing*, pages 356–365, 2005.
- [13] K. Munagala, S. Babu, R. Motwani, and J. Widom. The pipelined set cover problem. In *Proceedings 10th International Conference on Database Theory*, pages 83–98, 2005.
- [14] M. Queyranne. Structure of a simple scheduling polyhedron. *Math. Program.*, 58:263–285, 1993.
- [15] A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In *Proceedings 5th International Conference on Integer Programming and Combinatorial Optimization*, pages 301–315, 1996.

- [16] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, 1990.
- [17] R. Song, Z. Luo, J.-R. Wen, Y. Yu, and H.-W. Hon. Identifying ambiguous queries in web search. In *Proceedings 16th International Conference on World Wide Web*, pages 1169–1170, 2007.

## A Arbitrary Weight Vectors

In this section, we present details omitted from Section 2 of the paper.

### A.1 The weight reduction greedy algorithm fails in the arbitrary weights case

**Lemma A.1.** *The approximation guarantee of the weight reduction greedy algorithm is unbounded when the hyperedges have arbitrary weight vectors.*

**Proof.** Consider the graph that consists of  $k+1$  disjoint edges such that  $k$  of them have a weight vector of  $\langle 1, 0 \rangle$ , and one of them has a weight vector of  $\langle 0, w \rangle$ , where  $w$  is a large positive constant to be determined later. Let us analyze the performance of the greedy algorithm which selects, in every time step, a vertex that minimizes the overall weight contribution of the edges for the subsequent step. Notice that it selects at least one vertex incident on each of the edges having a weight vector of  $\langle 1, 0 \rangle$  before it chooses a single vertex of the extra edge. Therefore, the weight of its induced cover is at least  $(1 + \dots + k) + (k+2)w = \Omega(k^2 + kw)$ . On the other hand, an optimal algorithm may cover the extra edge first, and then attend to the edges having a weight vector of  $\langle 1, 0 \rangle$ . Accordingly, the weight of its induced cover is  $2w + (3 + \dots + (k+2)) = O(k^2 + w)$ . This implies that for a sufficiently large  $w$ , e.g.,  $w = \Omega(k^3)$ , the approximation guarantee of the greedy algorithm is  $\Omega(k)$ . Since one can utilize graph instances for which  $k \rightarrow \infty$ , the lemma follows. ■

### A.2 Proof of Lemma 2.2

Suppose the weight of some potential pair decreases. Accordingly, the width of the corresponding column in the histogram decreases. Clearly, the resulting histogram is monotonically non-decreasing. Moreover, when it is align to right boundary of the primary histogram, it must be contained in it. This holds since the primary histogram is monotonically non-decreasing. Similarly, if part of the weight of some potential pair is transferred to a potential pair with a smaller covering index then its effect is equivalent to transferring part of the width of some column of the histogram to an earlier position in that histogram. Again, it is easy to see that the resulting histogram is monotonically non-decreasing, and must be contained in the primary histogram. Figure 5 schematically describes these transformations.

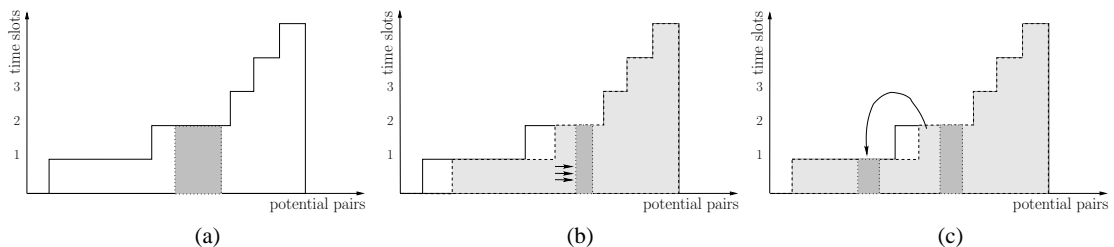


Figure 5: (a) The primary histogram; (b) The histogram which results by decreasing the width of some column; (c) The histogram that results by transferring part of the width of some column to an earlier position.

## B Non-Decreasing Weight Vectors

In this section, we present details omitted from Section 4 of the paper.

### B.1 Consistent ordering

**Definition B.1.** Let  $v$  and  $w$  be two vectors of equal length. We say that  $v$  and  $w$  have *consistent ordering* if there is a permutation  $\pi$  that sorts both  $v$  and  $w$ .

**Observation B.2.** Let  $v$  and  $w$  be two non-negative vectors of equal length  $\ell$ . The dot product of  $v$  and  $w$ , namely,  $v \cdot w = \sum_{i=1}^{\ell} v_i w_i$ , is maximal if their ordering is consistent.

**Proof.** Let  $v$  and  $w$  be two vectors that do not have consistent ordering. Consider a permutation  $\pi$  that sorts  $v$ , and let  $v' = \pi(v)$  and  $w' = \pi(w)$ . It is clear that  $v \cdot w = v' \cdot w'$ , and that  $w'$  is not sorted. In addition, there must be indices  $j, k$  such that  $j < k$ ,  $w'_j > w'_k$ , and  $v'_j < v'_k$ . One can reason about it by considering the complementary case in which  $v'_j = v'_k$ , for all indices  $j < k$  such that  $w'_j > w'_k$ , and noticing that an iterative sequence of swaps of the elements of  $w'$  determines another permutation that sorts both  $v$  and  $w$ . This stands in contradiction with the assumption that  $v$  and  $w$  do not have consistent ordering. Now, suppose we swap the elements of  $w'$  that reside in the  $j$ -th and  $k$ -th places. Essentially, this swap correlates to elements swap in  $w$ , which defines some new ordering  $\tilde{w}$ . Let us focus on  $\delta = v \cdot \tilde{w} - v \cdot w$ . Clearly,  $\delta = \pi(v) \cdot \pi(\tilde{w}) - \pi(v) \cdot \pi(w)$ . In addition, one can validate that  $\delta = (w'_j - w'_k) \cdot (v'_k - v'_j) > 0$ . This implies that any ordering that is not consistent can be improved. In particular, this proves that it is not maximal. ■