THE RANKABILITY OF DATA*

PAUL E. ANDERSON†, TIMOTHY P. CHARTIER‡, AND AMY N. LANGVILLE§

Abstract. This paper poses and solves a new problem, the *rankability problem*, which refers to a dataset's inherent ability to produce a meaningful ranking of its items. Ranking is a fundamental data science task. Its applications are numerous and include web search, data mining, cybersecurity, machine learning, and statistical learning theory. Yet little attention has been paid to the question of whether a dataset is suitable for ranking. As a result, when a ranking method is applied to an unrankable dataset, the resulting ranking may not be reliable.

The rankability problem asks: How can rankability be quantified? At what point is a dynamic, time-evolving graph rankable? If a dataset has low rankability, can modifications be made and which most improve the graph's rankability? We present a combinatorial approach to a rankability measure and then compare several algorithms for computing this new measure. Finally, we apply our new measure to several datasets.

Key words. ranking, rankability, linear program, integer program, combinatorial optimization, relaxation

AMS subject classifications. 90C08, 90C10, 52B12, 90C35

Code: https://github.com/IGARDS/rankability_toolbox

1. Introduction: Ranking vs. Rankability.

Ranking is a fundamental data science task that permeates almost every aspect of translating computational and algorithmic results into a form that a human can use. The objective in ranking is to sort objects in a dataset according to some criteria. We formulate ranking as a graph problem, finding the order or rank of vertices in a (weighted) directed graph. Ranking is hidden behind the question that asks "what's best?" "Best" involves quantifying, which then involves ranking. Amazon and Netflix ask: what's the best recommendation? Self-driving cars ask: what's the best route or lane? Other applications of ranking include resource allocation, web search, optimization, cybersecurity, and machine learning. Although the literature includes much research on ranking, what is lacking is significant research investigating particular foundational issues associated with ranking algorithms. In the absence of such foundational work, some investigators choose a ranking method and then accept the resulting ranking without asking questions such as: Can this ranking be trusted? Are certain parts of the ranking too similar to be disambiguated and, therefore, possibly meaningless? Such questions pertain to a topic that we call the *rankability of the data*. We study this analytics problem by posing the following questions, which are described further in Section 3: What is a dataset's degree of rankability and how should it be defined? Can a rankability measure be computed quickly? Are there subgraphs of the dataset that are rankable? Can we create a taxonomy of structured graphs according to their rankability? How can we modify a dataset to make it more rankable? Figure 1 gives an overview of the relationship between ranking and rankability.

We briefly summarize relevant findings for the *ranking problem*. Given information on pairwise relationships, the goal of ranking is to create a linear ordering of the items

*Submitted to the editors April 2018.

Funding: This work was funded by the —.

†Department of Computer Science, College of Charleston, SC 29401, USA (andersonpe2@cofc.edu).

‡Department of Mathematics and Computer Science, Davidson College, Davidson, NC (tichartier@davidson.edu).

§ Department of Mathematics, College of Charleston, SC 29401, USA (langvillea@cofc.edu).

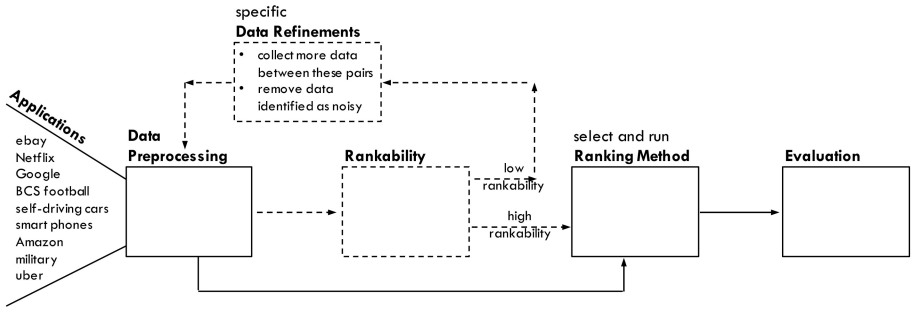

FIG. 1. ***Current Pipeline for Ranking vs. Proposed New Pipeline.*** *Ranking problems follow the pipeline shown in solid lines. This work proposes adding an important new step, the rankability step shown in dashed lines, which occurs prior to the computation of a ranking and measures how rankable the data is. If the data has low rankability, then our proposed work identifies which additional data to collect or remove (potential noisy data) in order to improve the rankability. Once the rankability measure is satisfactory, then a meaningful ranking that can be trusted is produced.*

that is consistent with the data. For this reason, ranking is also referred to as the *linear ordering problem* (LOP). The 2011 book by Reinelt and Marti [16] surveys the state of the art for the LOP. These authors describe the best approximate and exact algorithms for solving the LOP. Many heuristic methods and nearly all exact methods revolve around the so-called canonical LOP integer program (IP) shown below and its linear programming (LP) relaxation. The decision variable x_{ij} is binary and is 1 if item i is ranked above item j ($i > j$) and 0, otherwise. The objective function coefficients c_{ij} are the input data describing the results of a head-to-head matchup between items i and j. There are many ways to define c_{ij}. One from a sports context defines $c_{ij} = 55$ and $c_{ji} = 42$ if i beat j with the score 55 to 42.

$$\begin{aligned} \min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} & \\ x_{ij} + x_{ji} = 1 \qquad & 1 \le i < j \le n \\ x_{ij} + x_{jk} + x_{ki} \le 2 \qquad & i < j, i < k, j \ne k \\ x_{ij} \in \{0,1\} & \end{aligned}$$

The constraints above create the LOP polytope [25, 24] and much progress has been built around the theory related to this polytope, e.g., creating valid inequalities and cutting planes [11, 21, 22, 24].

We now attack a new problem, the rankability problem, that to our knowledge, we are the first to pose. This paper describes our approach to creating a rankability measure that answers the following questions.

1. How should rankability be defined?
2. Can such rankability measures be computed quickly?
3. Can rankable subsets of the data be identified?
4. Can a dataset's rankability be improved by adding or removing edges? Which edges?
5. Can a rankability measure be adapted for dynamic data?

We require that our proposed rankability measure satisfy three properties: (1) it must be **effective**, i.e., it must match our intuitive binary classification of structured datasets as rankable or unrankable; (2) it must be **efficient**, i.e., the time to compute the measure should be reasonable; and (3) it must be **algorithm-agnostic**, i.e.,

independent of a ranking or ranking method.

The third property separates our rankability work from related work on the quality of a ranking or ranking method. Some ranking methods produce both a ranking and a measure of the quality or confidence in that ranking [12, 15, 23, 3, 8, 2, 7, 4, 16, 24]. Such quality measures are not rankability measures. A quality of ranking measure $q = f(D, r)$ is a function of both the data D and the ranking r. In contrast, we desire a rankability measure $r = f(D)$ that is a function solely of the data. These three properties are precisely the same three properties required of clusterability measures. Clusterability is a new and active research area for clustering, another fundamental data task [1, 19].

We choose a few ranking applications to show what rankability enables.

- Our rankability work provides Netflix and Amazon with suggestions on exactly which pairwise comparisons to solicit from customers in order to most improve rankability, and thereby improve the quality of their rankings.
- Companies and even countries employ Analytic Hierarchy Process (AHP), a popular method for ranking alternative business and political decisions, to make million-dollar decisions. For example, AHP has been used to suggest resolutions to conflicts in South Africa, Northern Ireland, and Israel-Palestine. However, some data does not reduce to a linear ordering in the AHP sense [26]. Before making such high-stakes decisions, a rankability score can reveal how rankable the data is and provide suggestions as to which data to collect before a better ranking is computed.
- Phones and search engines present a ranking of auto-completion choices. To not irritate customers, auto-completion choices should only be presented when the list of choices is rankable.
- Rankability can be used to separate the signal from the noise–by identifying which votes in self-organizing trust systems (e.g., ebay or uber) are noise.

2. Related Work. Minimum violation rankings are one class of LOP methods [15, 23, 3, 8, 2, 7, 4, 16, 24]. These methods create an ordering that minimizes the number of upsets (or feedback arcs as they are called in the minimum feedback arc set problem [10]). Uparcs represent violations between the data and the ranking and manifest as nonzero elements in the lower triangular part of the matrix of dominance relations that has been symmetrically reordered according to the ranking. In the context of rankability, they are the y_{ij} variables of our rankability formulation presented in the next section. The y_{ij} variables are links that must be removed. Minimum violation work, however, does not consider link additions, the x_{ij} variables in our rankability formulation. Thus, our rankability work is more general than minimum violations work. Another way of summarizing the difference between our rankability work and minimum violations work: rankability considers both existing links and potential links, while minimum violations work considers only existing links. A further distinction: while minimum violation methods produce a score, the number of uparcs, that might be considered a partial rankability measure, it is not independent of the ranking and instead depends on how the ranking method defines violations. As mentioned above, we desire a rankability measure that is independent of a ranking or ranking method.

Another approach to rankability is sensitivity. Perhaps a graph is unrankable if it is highly sensitive to small changes in the graph. The sensitivity of linear systems and eigensystems, which are at the heart of most ranking methods, is well-studied [5, 6, 18]. Yet such sensitivity studies compare the original ranking to the perturbed

ranking and thus fail the independence property of a rankability measure. Our proposed rankability measure (Section 3) does incorporate ideas from sensitivity analysis. Traditional sensitivity analysis asks how small changes in the input affect the output. Instead, with rankability, we ask how *many* changes in the input are needed to create a *particular* output.

3. The Main Idea: rankability measure r. In this paper, we use data matrices and graphs interchangeably. A square matrix of data can be transformed into a graph and vice versa (e.g., with a weighted adjacency matrix or the normal form of a LOP matrix [16]). A rectangular matrix $\mathbf{A}$ of items by features can be transformed into a bipartite graph and vice versa. And this, if desired, can be transformed into a square matrix (e.g., $\mathbf{A}\mathbf{A}^T$). We first present our proposed rankability measure for unweighted (or uniformly weighted) graphs. Later in Section 4.1, we relax this restriction and show how to extend our rankability measure to weighted graphs.

Figure 2 shows our binary classification of classes of several artificial graphs with special structure. Our rankability work begins with the question: what is the difference between the graphs at the extremes of Figure 2: the dominance graph, which intuitively seems very rankable, and the completely connected graph, which seems unrankable? The dominance graph has exactly **one unquestionable** ranking of its nodes. On the other hand, any ranking for the completely connected graph is open to debate. Perhaps one or more dominance graphs are hidden in the completely connected graph. Clearly, a dominance graph can only be recovered from a completely connected graph if perturbations are allowed. This prompts two key questions for any graph whose answers create the rankability measure.

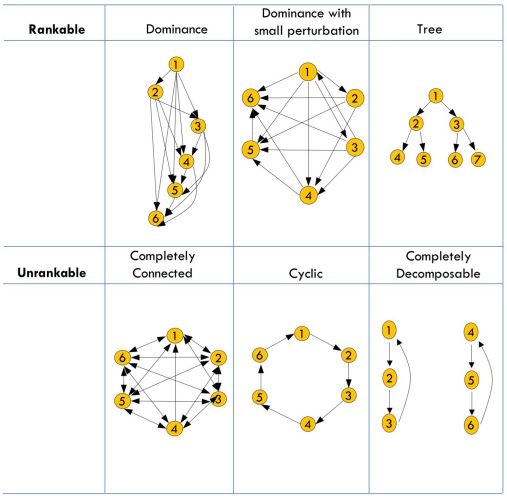

FIG. 2. ***Structured artificial graphs*** *grouped as rankable or unrankable.*

(1) *Distance from ideal:* How far is the graph from a dominance graph? More precisely, what is the minimum number of changes (edges added or removed) that must be made to the completely connected graph to transform it into a dominance graph? We denote the answer to this question as k.

(2) *Uniqueness:* Given that it takes k changes to transform the original graph into a dominance graph, how many different dominance graphs are this distance from the original graph? We denote the answer to this second question p and let P be the corresponding set of rankings so that $p = |P|$.

We ask and answer these questions for any graph, creating a rankability measure r such that $0 \leq r = 1 - \frac{kp}{k_{max}p_{max}} \leq 1$, where $k_{max} = (n^2 - n)/2$ is the maximum number of changes that can be made to an n-node graph and $p_{max} = n!$ is the maximum number of rankings of an n-node graph. Consider the following two $\mathbf{D}$ matrices for 4-node graphs.

$$\mathbf{D}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \mathbf{D}_2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The dominance information in $\mathbf{D}_1$ shows that node 1 beat node 2 who beat node 3 who beat node 4, which creates the ranking $1 > 2 > 3 > 4$. $\mathbf{D}_2$ has this same information plus a few additional dominance relations (1 also beat 3 and 2 also beat 4) that add more confidence to the ranking $1 > 2 > 3 > 4$. This is reflected in the rankability scores: $r(\mathbf{D}_1) = 1 - \frac{3*3}{6\times24}$ and $r(\mathbf{D}_2) = 1 - \frac{1}{6\times24}$. Because $r(\mathbf{D}_2) > r(\mathbf{D}_1)$, $\mathbf{D}_2$ has better rankability than $\mathbf{D}_1$.

Figure 3 shows our rankability measure on several small $n = 6$ artificial graphs. Notice that r meets two of our three desirable properties for a rankability measure.

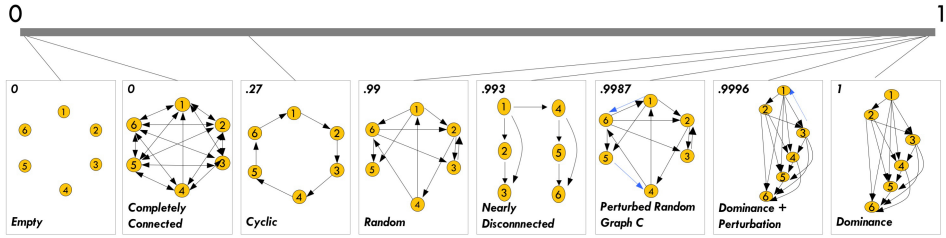

FIG. 3. ***Rankability scores*** *r, shown in the upper left corner of each box, for several* $n = 6$ *graphs.*

First, it is effective and matches our intuitive hunches about rankable versus unrankable graphs. Second, it is independent of a ranking or ranking method and instead uses only the structure of the graph to create a score. We discuss the remaining property, efficiency, in Section 5.

3.1. Example with $n = 6$ graph. Figure 4 shows an $n = 6$ example with k and p for a perturbed dominance graph. (Later in Section 4 we explain how k and p are computed.) The ranking of [1 2 3 5 4 6] reorders the matrix of pairwise rela-

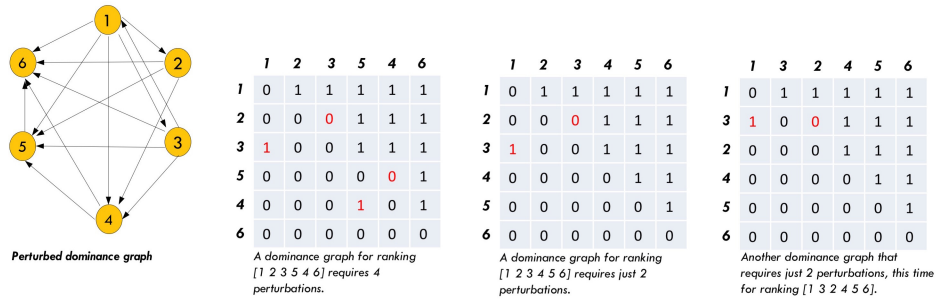

FIG. 4. ***Perturbed dominance graph*** *for three rankings with the changes of link additions and deletions highlighted in red. The red 1's below the diagonal indicate links that should be removed while the red 0's above the diagonal indicate links that should be added in order to transform this graph to a dominance graph for the intended ranking. The number of red elements indicates the number of perturbations. For this graph,* $k = 2$ *since this is the minimum number of perturbations needed to transform the graph to a ranking, i.e., a dominance graph. And* $p = 2$ *since there are two rankings, the two rightmost, that require just two perturbations.*

tionships and reveals that four changes must be made in order to transform this matrix into a perfect dominance graph. The two nonzeros in the lower triangular part of the reordered matrix represent two edges that must be removed while the two zeros in the upper triangular part represent two edges that must be added. Yet a ranking with a smaller number of perturbations exists. The ranking of [1 2 3 4 5 6] requires only two changes, which is the minimum for this graph, meaning $k = 2$. This is not the only $k = 2$ ranking. The only other ranking with $k = 2$ is [1 3 2 4 5 6], which means that $p = 2$ and the set P consists of these two rankings, [1 2 3 4 5 6]

and [1 3 2 4 5 6].

3.2. Using the set P of rankings to improve rankability. The set P contains useful information as the example of Figure 5 shows with an $n = 6$ random graph with $k = 9$ and $p = 12$. The rankings in P can be used to create three matrices, $\mathbf{P}_a$, $\mathbf{P}_d$, and $\mathbf{P}_>$, which summarize the information, respectively, about adding links, deleting links, and dominance relations. $P_a(i, j)$ gives the percentage of rankings in P requiring a link be added from i to j. $\mathbf{P}_d(i, j)$ gives the percentage of rankings in P requiring a link be deleted from i to j. $\mathbf{P}_>(i, j)$ gives the percentage of rankings in P having $i > j$. For this $n = 6$ example, all twelve of the rankings in P added a link from 4 to 2 and removed the link from 2 to 4. This suggests that the link from 2 to 4 may be *noise* as it is inconsistent with every ranking of the nodes.

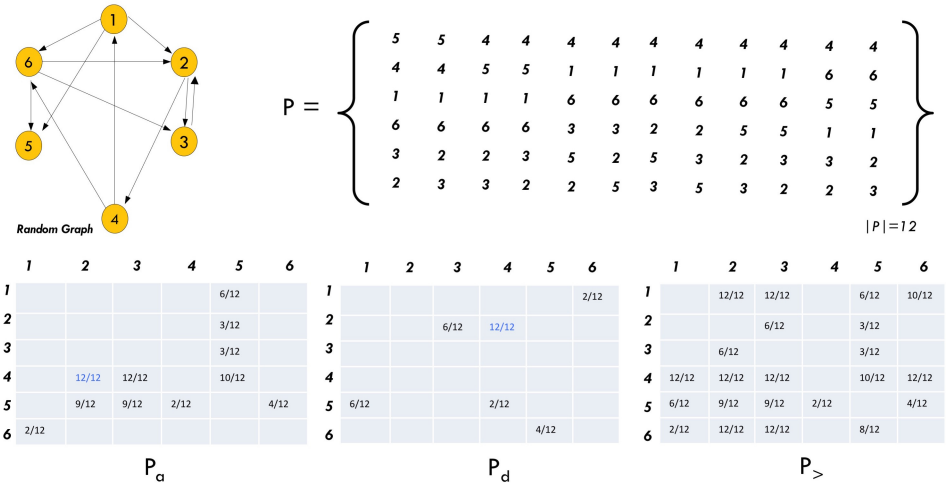

FIG. 5. ***Random graph with full P set and three matrices, $\mathbf{P}_a$, $\mathbf{P}_d$, and $\mathbf{P}_>$, built from P.*** *Each column in P is a ranking. The matrices $\mathbf{P}_a$, $\mathbf{P}_d$, and $\mathbf{P}_>$ summarize information, respectively, about adding links, deleting links, and dominance relations. Notice that $P_a(4, 2) = 12/12$ and $P_d(2, 4) = 12/12$, which means that all twelve rankings suggest adding a link from 4 to 2 and removing the link from 2 to 4. Thus, the existing link from 2 to 4 may be noise.*

The information in $\mathbf{P}_a$, $\mathbf{P}_d$, and $\mathbf{P}_>$ can be used to make recommendations on how to improve a graph's rankability. For example, $\mathbf{P}_>$ shows that there is maximal indecision about whether $3 > 2$ or $2 > 3$, since $P_>(3, 2) - P_>(2, 3) = 0$ with half of the rankings in P having $3 > 2$ and the other half having $2 > 3$. Another indicator of ambiguity in the rankings between two items is the maximum spread in their rank positions. For example, the deviation in rank positions between items 5 and 3 varies from 1 (in the ranking [4 1 6 5 3 2]) to 5 (in the second ranking in P [5 4 1 6 2 3]). The ambiguity of individual items or groups of items can also be tracked. Item 5 ranges from first rank position to last whereas item 4 holds first position in 10 of the 12 rankings in P and second in the remain-

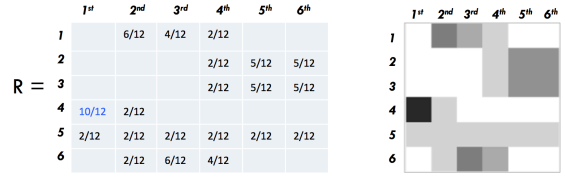

FIG. 6. ***Summary of Rank Information in P.*** *Element (i, j) of $\mathbf{R}$ is the percentage of rankings in the set P that rank item i in the j^{th} rank position. The pixel plot summarizes this information and, for larger graphs, can provide a quick overview of both confidence and ambiguity in the ranking. The darker the pixel the more agreement that the item belongs in that rank position. In this example (Random graph), item 4 deserves a spot at the top of the ranking and items 2 and 3 belong at the bottom.*

ing two rankings. Thus, we can create summaries of rank information (Fig. 6) and confidence measures on the rank positions of items.

Figure 7 uses such heuristic rules to make two changes to the original random graph to create several perturbed random graphs. Observe the changes in the rankability scores, as shown by the decreasing size of P. Having the full set P is ideal,

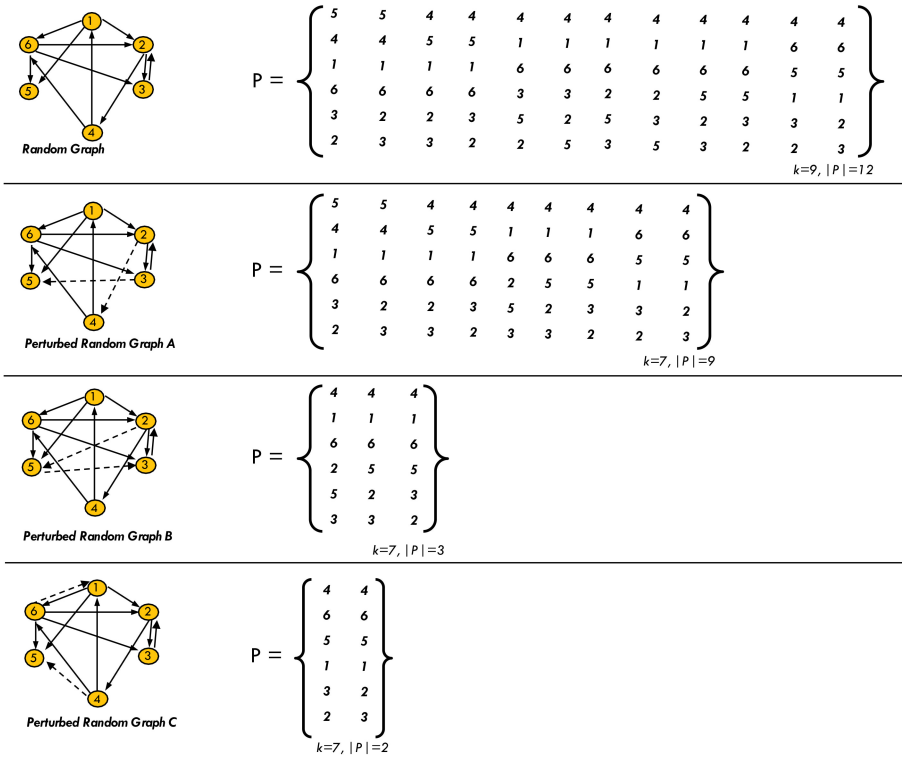

FIG. 7. ***Improving a graph's rankability with link perturbations.*** *The $n = 6$ example graph at the top, a random graph, is perturbed three different times, each time by making two link perturbations, indicated by dashed lines. The link perturbations are made according to the heuristics described on the previous page. The rankability measure improves from graph A to B to C since the value of k reduces from 9 to 7 and the size of the P set successively shrinks.*

but even a partial set of members of P gives rankability information. In particular, if members of partial P are diverse and vary wildly, then $|P|$ is likely large and the graph less rankable than a partial P whose members are similar. The diversity of a set of members of partial P can be measured by the average pairwise rank correlation with Kendall or Spearman measures [13, 14].

4. Theory: Rankability Integer Programs to find k and P. The next section discusses some algorithms for computing k and p, all of which involve the *rankability integer programs* (IP) described in this section. In all cases, the input is a matrix $\mathbf{D}$ where d_{ij} is 1 if a link exists from item i to item j, meaning $i > j$ and 0, otherwise.

4.1. Original Formulation. This formulation, referred to as the original formulation, has two sets of decision variables, x_{ij} and y_{ij}, that give information about which links should be added or deleted to transform $\mathbf{D}$ into a dominance graph. The

decision variable x_{ij} is 1 if a link is added from i to j, and 0, otherwise. The decision variable y_{ij} is defined similarly for the removal of a link from i to j.

$$\min \sum_{i \neq j} (x_{ij} + y_{ij})$$

$$(d_{ij} + x_{ij} - y_{ij}) + (d_{ji} + x_{ji} - y_{ji}) = 1 \quad \forall\, i < j \qquad \text{(anti-symmetry)}$$

$$(d_{ij} + x_{ij} - y_{ij}) + (d_{jk} + x_{jk} - y_{jk}) + (d_{ki} + x_{ki} - y_{ki}) \leq 2 \quad \forall\, j \neq i, k \neq j, k \neq i \qquad \text{(transitivity)}$$

$$\sum_{i \neq j} (x_{ij} + y_{ij}) \leq k_{init} \qquad \text{(smart initialization)}$$

$$x_{ij} + y_{ij} \leq 1 \quad \forall\, i, j \qquad \text{(add or remove or neither)}$$

$$0 \leq x_{ij} \leq 1 - d_{ij} \quad \forall\, i, j \qquad \text{(only add potential links)}$$

$$0 \leq y_{ij} \leq d_{ij} \quad \forall\, i, j \qquad \text{(only remove existing links)}$$

$$x_{ij}, y_{ij} \in \{0, 1\} \quad \forall\, i \neq j \qquad \text{(binary)}$$

This formulation has $2n^2 - 2n$ variables and $n^3 - \frac{3}{2}n^2 + \frac{3}{2}n + 1$ main constraints plus $2n^2$ lowerbound and upperbound constraints that fix many variables immediately. The anti-symmetry and transitivity constraints appeared earlier in the canonical IP formulation of the LOP. In this case, they force the perturbed matrix $\mathbf{D} + \mathbf{X} - \mathbf{Y}$ to be a dominance matrix. A dominance matrix is simply the matrix representation of a dominance graph and therefore can be symmetrically reordered to strictly upper triangular form. The ordering of nodes that achieves this upper triangular form is the ranking. The objective function value gives k, which is the minimum number of perturbations to $\mathbf{D}$ (link additions in $\mathbf{X}$ and link deletions in $\mathbf{Y}$) required to achieve a dominance graph. The number of optimal extreme point solutions to this rankability IP is p and the set of optimal extreme point solutions is P. Finding all optimal (extreme point) solutions is known to be a difficult problem and thus computing the p part of the rankability measure requires some algorithmic ingenuity, which is discussed in Section 5.

Our rankability model above deals with unweighted (i.e., binary weighted) graphs, the first simplest case. The extension to weighted graphs is straightforward: costs can be added to the objective function of the rankability IP. There are several possible implementations. (1) For a weighted graph with input matrix $\mathbf{C}$, the objective function becomes $\min \sum_i \sum_j x_{ij} + \sum_i \sum_j c_{ij} y_{ij}$. In this way, the removal of a link with a heavy weight penalizes the objective function. (2) Similarly, costs can also be added to the $\mathbf{X}$ matrix part of the objective function of the rankability IP. In this case, two cost matrices may be given: the matrix $\mathbf{C}^+$ contains the costs c^+_{ij} of adding a link from i to j and the matrix $\mathbf{C}^-$ contains the costs c^-_{ij} of removing a link from i to j. Hence, the objective function becomes $\min \sum_i \sum_j c^+_{ij} x_{ij} + \sum_i \sum_j c^-_{ij} y_{ij}$. We expect this modification to be of interest in military applications where, for example, addition, i.e., construction, of a link in a communication network may carry a different weight than removal of the same link. (3) The measure of perfection can be c_{max} times the perfect dominance graph where c_{max} is the largest element in $\mathbf{C}$. In this case, the objective function becomes $\sum_i \sum_j [c_{max} - d_{ij}(c_{max} - c_{ij})] x_{ij} + [c_{ij} - d_{ij}(c_{max} - c_{ij})] y_{ij} + d_{ij}(c_{max} - c_{ij})$.

One caution: when dealing with weighted graphs, it is important that the constraints of the mathematical program use the binary values of the input matrix $\mathbf{D}$ and not the weighted values in the cost matrices $\mathbf{C}$. In other words, for the weighted graph $\mathbf{C}$, the binary matrix $\mathbf{D} = \mathbf{C} > \mathbf{0}$ must be formed where $d_{ij} = 1$ if $c_{ij} > 0$ and 0, otherwise.

4.2. Alternative Formulation 1. The original formulation can be significantly simplified by defining $z_{ij} = d_{ij} + x_{ij} - y_{ij}$. This creates the first alternative formulation shown below. Note that this formulation is a maximization problem.

$$\begin{aligned}
&\max \sum_{i \neq j} d_{ij} z_{ij} \\
&z_{ij} + z_{ji} = 1 \quad \forall\, i < j && \text{(anti-symmetry)} \\
&z_{ij} + z_{jk} + z_{ki} \leq 2 \quad \forall\, j \neq i, k \neq j, k \neq i && \text{(transitivity)} \\
&z_{ij} \in \{0, 1\} \quad \forall\, i \neq j && \text{(binary)}
\end{aligned}$$

The constraints of this new formulation encompass those of the original formulation and are arrived at with simple substitution $z_{ij} = d_{ij} + x_{ij} - y_{ij}$. Maximization of the objective function of the new formulation occurs when both $d_{ij} = 1$ and $z_{ij} = 1$, which means $x_{ij} = 0$ and $y_{ij} = 0$. This is equivalent to the original formulation that desires to minimize the values in x_{ij} and y_{ij}. The following rules are used to translate the solution from this alternative formulation into the solution for the original formulation. If $z_{ij} = 0$ and $d_{ij} = 1$, then set $y_{ij} = 1$. If $z_{ij} = 1$ and $d_{ij} = 0$, then set $x_{ij} = 1$. Then k is the number of nonzeros in $\mathbf{X}$ plus the number of nonzeros in $\mathbf{Y}$, i.e., $k = nnz(\mathbf{X}) + nnz(\mathbf{Y})$.

As with the original formulation, weighted graphs can be handled by adding weights to the objective function. However, with this formulation, because there in only one decision variable per link $(i.j)$ rather than two, one for link addition and one for link removal, the ability to apply different weights to the two operations of link addition and link removal is lost.

4.3. Alternative Formulation 2. This alternative formulation classifies decision variables as either Type 1 or Type 2 and exploits this classification. The decision variable z_{ij} is 1 if link (i, j) is changed. The decision variable z_{ij} associated with link (i, j) is defined as a Type 1 variable if the dominance relation between items i and j is indeterminate, i.e., $d_{ij} = d_{ji}$. To transform the data into a ranking, this tie must be broken by the integer program so either $z_{ij} = 1$ or $z_{ji} = 1$. The decision variable z_{ij} associated with link (i, j) is defined as a Type 2 variable if a dominance relation between items i and j exists, i.e., $d_{ij} = 1 - d_{ji}$. In this case, the integer program determines that either the dominance relation remains as it is (so that $z_{ij} = 0$ and $z_{ij} = 0$) or it flips (so that $z_{ij} = 1$ and $z_{ij} = 1$). A strictly upper triangular matrix $\mathbf{K}$ is formed from $\mathbf{D}$ where $k_{ij} = 1$ if $d_{ij} = d_{ji}$ and $k_{ij} = 2$ if $d_{ij} + d_{ji} = 1$ for $i < j$. Let n_1 be the number of Type 1 variables, i.e., the number of ones in $\mathbf{K}$. Let n_2 be the number of Type 2 variables, i.e., the number of twos in $\mathbf{K}$.

$$\begin{aligned}
&\max \sum_{i<j} (2(k_{ij} - 1)) z_{ij} \\
&z_{ij} + (3 - 2k_{ij}) z_{ji} = 2 - k_{ij} \quad \forall\, i < j && \text{(anti-symmetry)} \\
&(1 - 2d_{ij}) z_{ij} + (1 - 2d_{jk}) z_{jk} + (1 - 2d_{ki}) z_{ki} \leq 2 \quad \forall\, j \neq i, k \neq j, k \neq i && \text{(transitivity)} \\
&z_{ij} \in \{0, 1\} \quad \forall\, i \neq j && \text{(binary)}
\end{aligned}$$

The following rules are used to translate the solution from this alternative formulation into the solution for the original formulation. If $z_{ij} = 1$ and $d_{ij} = 1$, then set $y_{ij} = 1$. If $z_{ij} = 1$ and $d_{ij} = 0$, then set $x_{ij} = 1$. Then k is n_1 plus the objective function value of this integer program. Weighted graphs can be handled by appropriately adding weights to the objective function.

This formulation has a few advantages. First, Type 1 variables reveal that n_1 is a lowerbound on k. Second, Type 2 variables reveal that k values can only increase from n_1 in increments of 2. Third, unlike the previous formulations, both the constraints and the objective function have differentiation in their coefficients. These properties are exploited by the next section's algorithms for finding p and the set P.

All three formulations require $O(n^2)$ variables and $O(n^3)$ constraints. Each formulation described how to produce k from the optimal objective value. And the set P consists of all optimal solutions to each formulation. Further, LP relaxations of each formulation may be executed instead. We refer to the feasible region defined by the constraints of a LP relaxation of a formulation as the *LP rankability polytope*. We call the convex hull of the feasible solutions of the integer program the *IP rankability polytope*.

LEMMA 4.1. *If the LP relaxation of any rankability IP formulation results in an* integer *objective value, then this objective value is optimal for the integer program.*

Proof. (By contradiction.) Assume otherwise. That is, assume the LP relaxation of a rankability IP formulation results in a fractional, non-integer, objective value k that is not optimal for the corresponding integer program. Without loss of generality, assume the formulation is the original rankability formulation of Section 4 whose objective function is a minimization problem. Then means that the objective value of the integer program $f(\mathbf{x}_{IP})$ must be less than k so $f(\mathbf{x}_{IP}) < k$. Yet the LP, being a relaxation, must have $f(\mathbf{x}_{LP}) \leq f(\mathbf{x}_{IP}) < k$. But $f(\mathbf{x}_{LP}) < k$ contradicts the assumption that $f(\mathbf{x}_{LP}) = k$. ∎

When the hypothesis of Theorem 4.1 holds (i.e., the LP relaxation of the rankability IP has an *integer* objective value), the following statements are true. The *hyperplane for the optimal face* of the LP polytope is the hyperplane for the optimal face of the IP polytope. However, the optimal faces may differ. The optimal face of the IP is either the optimal face of the LP or is contained within the LP optimal face. While the binary extreme points are shared by both optimal faces, the LP optimal face *may* have additional extreme points, fractional extreme points.

Our experiments show that alternative formulation 2 is repeatedly the fastest of the three formulations. Thus, Table 1 reports the runtimes for solving this formulation, resulting in the optimal value k and at least one member of P. The mean and 95% lower and upper confidence intervals are calculated over 100 graphs that were generated with random and increasing perturbations from the dominance graph. For example, an $n = 20$ dominance graph has $(n^2 - n)/2 = 190$ links and we removed 0%, 15%, 30%, 45%, 60%, and 75% of these links to create "Dominance + Perturbation" graphs. Thus, all these graphs have good rankability scores. Notice that these rankable graphs solve quickly. A rankable graph of size $n = 100$ generally solves the integer program in about a minute and a half. On the other hand, unrankable graphs that were generated similarly (i.e., making random perturbations instead to the completely connected graph) take much longer. An $n = 50$ unrankable graph took 15 minutes and an $n = 100$ unrankable graph took 45 minutes. These experiments were run in Gurobi Optimization on a single node with 2 CPUs, 24 cores, and 128 GB of memory.

Graphs of up to $n = O(10^3)$ are within reach with a standard technique in optimization, known as constraint relaxation. The transitivity constraints of the integer programs of this section create $O(n^3)$ constraints, which limit the size of graphs that can be handled. Constraint relaxation is an iterative procedure that works as follows. Initially solve the problem with no transitivity constraints. Find the transitivity con-

TABLE 1
Runtimes of Alternative Formulation 2

n	rankable+perturbation graphs 95% lower	mean	95% upper
10	3.2s	3.3s	3.4s
25	4.2s	4.4s	4.5s
50	13.7s	14.0s	14.4s
75	39.5s	40.3s	41.1s
100	95.5s	97.5s	99.5s

straints violated by this initial solution, add these to the problem, and solve this modified problem. Repeat, adding violated transitivity constraints at each iteration, until no transitivity constraints are violated. This solution is then optimal for the rankability IP. Our experiments with rankability problems with $100 < n < 500$ show that constraint relaxation uses much less than half of one percent of the $O(10^5)$ transitivity constraints.

5. Algorithms for computing r. We divide our rankability algorithms into two classes: exact methods and approximate methods.

5.1. Exact Methods. The integer programs of the previous section find the exact value for k and one (or several, depending on the settings of the optimization solver) members of the set P. For each formulation, the set P is the set of all optimal solutions and the cardinality of this set is required for our definition of rankability from Section 3. In general, finding all optimal solutions of an integer or linear program is known to be a difficult problem. Thus, some ingenuity is required to create our rankability measure.

We first discuss related work, the Double Description method [20], which, given a set of constraints describing a finite polytope, enumerates all of its extreme points (or vice versa). We begin by solving the linear relaxation of one of the IP formulations of the previous section arriving at the objective value k that, if integer, is optimal for the integer program. When optimal, this value of k is used to add to the feasible region a constraint representing the optimal face. Thus, with this additional constraint the polytope represents the optimal face. Since P consists of all binary extreme points on the optimal face of the linearly relaxed rankability polytope, we input to the Double Description method the rankability constraints from the formulation plus the constraint for the optimal face. The method outputs the extreme points on the optimal face. We discard any fractional ones, keeping the rest as members of P. Even with Fukuda's efficient implementation [9] of the Double Description method, this is an expensive process and limited to graphs of small size, $n \approx 10$.

We now describe a more practical and scalable method for finding the full set P, which we refer as *Parallel Exhaustive Enumeration with Pruning*. This algorithm produces the exact answer to a rankability problem (k, p, and the full set P) by considering, but not enumerating, all $n!$ rankings, one at a time or **in parallel**. Because enumerating and evaluating all $n!$ rankings is only practical for $n \approx 10$, we use two conditions to efficiently prune branches of much larger trees of rankings, enabling the exact solution of graphs of $n \approx O(10^2)$. Pruning condition 1 eliminates a branch of the tree of rankings if the fitness score (i.e., k) of the corresponding **subranking** is greater than the best (or optimal) value of k found thus far. Pruning condition 2 elim-

inates a branch if the subranking violates the anti-symmetry constraint of alternative formulation 2 of Section 4.3.

The following $n = 4$ example demonstrates the two pruning conditions.

$$\mathbf{D} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}, k^* = 1,\ p^* = 1, \text{ and } P = [1\ 2\ 3\ 4]^T.$$

And the matrix $\mathbf{K}$ associated with alternative formulation 2 of Section 4.3, which is required for pruning condition 2, is

$$\mathbf{K} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} & 2 & 1 & 2 \\ & & 2 & 2 \\ & & & 2 \\ & & & \end{pmatrix} \end{array}.$$

Figure 8 shows the tree of the $n!$ rankings that must be considered by the exhaustive enumeration method. The two pruning conditions can be checked at various levels of the tree. The figure shows pruning applied to this $n = 4$ example at the second level of the tree. Of course, it is very advantageous when a pruning condition is met near the top or stem of the tree.

For this example, because the known optimal fitness score is 1 (since $k^* = 1$), pruning condition 1 eliminates any branches below a subranking with a fitness score of 2 or more. For example, branches below the subranking of $[2\ 1]^T$ are pruned because the submatrix of $\mathbf{D}$ associated with this subranking

$$\mathbf{D} = \begin{array}{c} \\ 2 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 2 & 1 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \end{array}$$

has a fitness score of 2 (i.e., it is two violations from the perfect 2×2 dominance graph in strictly upper triangular form) and this is more than $k^* = 1$.

Pruning condition 2 checks if pairs (i, j) for $i < j$ in the subranking satisfy the anti-symmetry constraint from alternative formulation 2 from Section 4.3, namely, $z_{ij} + (3 - 2k_{ij})z_{ji} = 2 - k_{ij}$, where the $\mathbf{Z}$ submatrix is formed by subtracting the corresponding submatrix of $\mathbf{D}$ from the perfect dominance graph of matching size. As soon as one (i, j) pair in the subranking is found that violates this anti-symmetry constraint, stop and prune all branches below this point in the tree. For this example, pruning condition 2 eliminates any branches below the subranking of $[1\ 3]^T$ because the anti-symmetry constraint requires $z_{13} = z_{31}$ yet $z_{13} = 0$ and $z_{31} = 1$, where the submatrix $\mathbf{Z}$ is the 2×2 perfect dominance submatrix minus the 2×2 submatrix of $\mathbf{D}$.

$$\mathbf{Z} = \begin{array}{c} \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \end{array} - \begin{array}{c} \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array} = \begin{array}{c} \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 3 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \end{array}.$$

The tree of $n!$ rankings for the exact method can be traversed in either a breadth first or a depth first manner. Breadth first traversal enables a parallel implementation. Early termination of a breadth first traversal of the exact method results in an

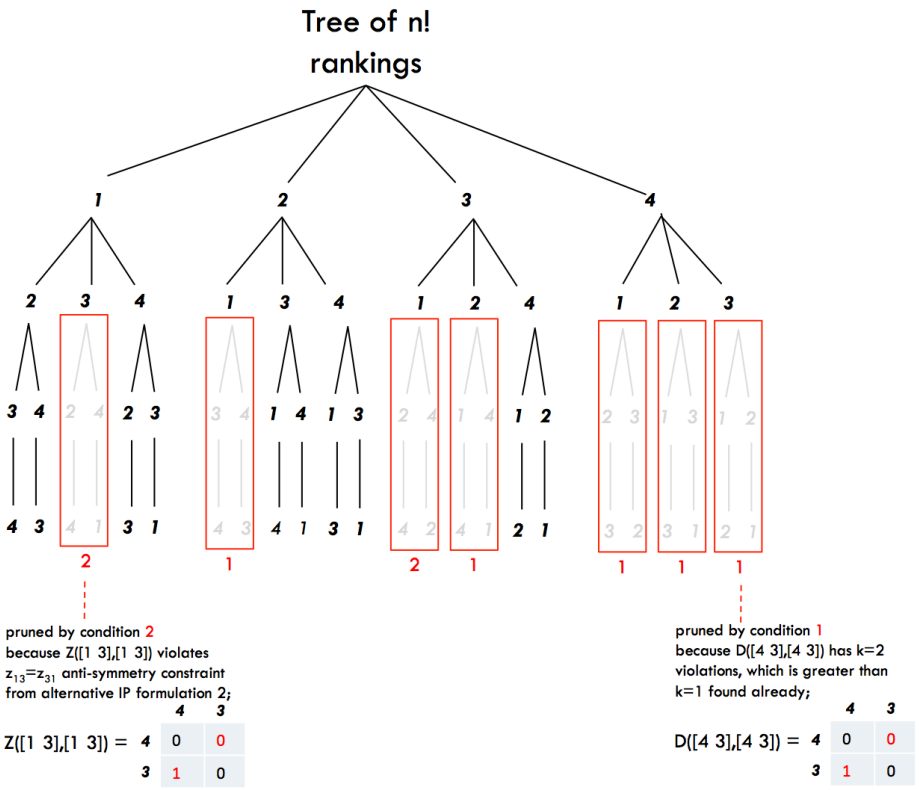

FIG. 8. ***Pruning conditions with tree of*** $n!$ ***rankings.*** *Pruning is applied at the second level of this* $n = 4$ *tree to eliminate 14 of the 24=4! rankings from consideration.*

upperbound p_{ub} on $p = |P|$ where p_{ub} is $n!$ minus the number of pruned leaves at termination. For a numerical example, the $n = 32$ graph for the 2016 NFL football season creates a tree of $32! = 2.6 \times 10^{35}$ rankings. Tables 2 shows the percentage of rankings remaining to be considered after various early termination points. Notice that the pruning algorithm makes progress quickly in the beginning, then as the easy prunings have been made, the algorithm must traverse deeper down the branches of the tree which requires more processing. The last 18 hours of processing are required to consider the final .000001% of the rankings, yet because 32! is so large, this small percentage is still over 10^{27} items. At 6.5 hours, the p_{ub} value is 10^{27}, which is much larger than the exact value of p for this example, $p = 9,305,550 \approx 9 \times 10^6$.

TABLE 2
Progress of Exact Pruning Algorithm for $n = 32$ *NFL example*

time	% remaining
.5h	64%
1h	31%
2h	<1%
4.5h	<.01%
6.5h	<.000001%
23h	0%

On the other hand, early termination of a depth first traversal of the exact method results in a partial set P and thus provides a lowerbound p_{lb} on p where p_{lb} is given by the cardinality of the partial P. Both the runtime and the size of the set P (i.e., $p = |P|$) depend on the number of prunings of the exact method. A rankable graph has a small set P and thus many prunings, which results in a fast runtime. For example, an $n = 20$ "Dominance + 75% Perturbation" graph takes 4.48 minutes to run the exact method and produce an exact p of 1020. On the other hand, an unrankable graph has a large set P and few prunings, resulting in a long runtime. For example, an $n = 20$ "Connected + 75% Perturbation" graph takes 26.33 minutes to run the exact method, producing an exact p of $133,460$. In this way, the runtime of the exact method gives an indication of a graph's rankability. Suppose that after 3 hours the exact method is terminated early for a particular graph. This can be compared to the runtime of the worst case graph, the completely connected graph, which took, say, 5 hours for full termination traversing the entire graph.

5.2. Approximate Methods. This section describes three approximate methods for rankability. The first two methods are approximate because, rather than finding all members of P, they find just a subset of P, a partial P denoted $\hat{P}$. The final method is approximate because it summarizes approximately the information in P without actually forming P.

The first approximate method, *Parallel IP runs with Random Reorderings*, works as follows. Select one of the rankability IP formulations of Section 4 and set the IP solver to collect multiple optimal solutions as it proceeds (e.g., with Gurobi's PoolSearch routine). Then reorder the rows and columns of the input matrix $\mathbf{D}$ according to a random ranking and run the IP solver with PoolSearch again. This run collects more members of P. The idea is that a different reordering of $\mathbf{D}$ starts the IP branch and bound tree with a different initial feasible solution and thus randomness makes the solver work from another side of the polytope. These IP runs are done in parallel and the union of the PoolSearch results is taken. While the resulting set is not guaranteed to be the full set P and is only a partial P denoted $\hat{P}$, it contains a representative and diverse collection of members of P. The user can control the size of the partial $\hat{P}$ by how many reorderings are used. Because a partial set $\hat{P}$ is produced, this approximate method provides a lowerbound p_{lb} on p where p_{lb} is is given by the cardinality of the partial $\hat{P}$.

The second approximate method is *Early Termination of the Parallel Exhaustive Enumeration with Pruning and Depth First Traversal*, the exact method described above in Section 5.1. The user can limit the runtime of the exact method and thereby control the size of the partial P set. Suppose that after one hour the exact method is terminated before the entire tree has been explored. Then the resulting set P is partial. Allowing a longer runtime before termination results in a larger, though still partial, set $\hat{P}$.

The third and final approximate method is the *LP Interior Point Solution*. This method solves the linear relaxation of one of the IP formulations of the previous section by an interior point method. If a linear program (LP) has multiple optimal solutions, an interior point method such as the primal-dual interior point method of Mehrotra and Ye converges to an optimal solution that is near (or is) the centroid of the optimal face [17]. For our rankability problem, if the objective value of the LP relaxation of one of our rankability IPs is integer, then this objective value is optimal for the integer program and the LP optimal solution found by the interior point method is near the centroid of the IP optimal face. Because it is a LP relaxation,

the extreme points of the optimal face are not limited to binary extreme points. Fractional extreme points on the optimal face may exist. Thus, the LP interior point solution is a convex combination of these binary and fractional extreme points. On the other hand, the ideal centroid, the centroid of the optimal face of the IP polytope, is a convex combination of only the binary extreme points. Thus, the LP centroid solution approximates the ideal centroid of the exact full set P of binary extreme points on the optimal face of the IP polytope. Fortunately, the LP centroid solution well approximates the ideal centroid. With the Double Description method we have verified for small n that fractional extreme points are extremely rare on the optimal face of the LP polytope. Further, the numerical experiments of the supplement and the next section support this conclusion for larger n. Being a convex combination of all extreme points, the centroid can be considered a summary of the information in the individual extreme points. And this summary enables the approximation of the matrices $\mathbf{P}_a$, $\mathbf{P}_d$, $\mathbf{P}_>$, which we showed earlier in Section 3.2 can be used to make suggestions for which links to add or remove to improve the graph's rankability.

THEOREM 5.1. *If the objective value of the LP relaxation of one of the rankability IPs from Section 4 is integer, then the optimal face of the rankability LP has $|P|$ binary extreme points (and thus, $\exists$ $|P|$ optimal solutions to the IP) and each corresponds to a member of P.*

Proof. Our direct proof consists of four steps: (1) we show that the hyperplane of the optimal face for the LP rankability polytope is the same as that for the IP rankability polytope, (2) we show that each member of P is an optimal solution for the rankability LP, (3) we show that each member of P corresponds to a binary solution of the rankability LP, and (4) we show that each member of P is an extreme point of the polytope of the rankability LP. STEP 1: By Lemma 4.1 we know that the integer objective value of the LP relaxation of the rankability IP is also optimal for the IP. Thus, the hyperplane of the optimal face for the LP rankability polytope is the same as that for the IP rankability polytope. STEP 2: By definition, a member of P is a ranking and thus satisfies the anti-symmetry and transitivity constraints of the rankability LP. (Without loss of generality, we use the original formulation of the rankability problem.) Each member of P is k link perturbations away from the original graph. Link perturbations are binary (either a link is added or removed) and not fractional. Thus, the constraint $x_{ij} + y_{ij} \leq 1$ is satisfied. Similarly, the lowerbound and upperbound constraints are also satisfied. Thus, each member of P is feasible for the rankability LP. Further, each member of P is optimal for this LP since the objective function value is k. Thus, each member of P is an optimal solution on the optimal face of the rankability polytope. STEP 3: Each member of P is a ranking. When the rows and columns of the input matrix $\mathbf{D}$ are symmetrically reordered according to the ranking, a zero in the (i, j) position of the upper triangular of this reordered matrix represents a link that must be added to create a dominance graph, i.e., $x_{ij} = 1$. A nonzero in the (i, j) position of the lower triangular of this reordered matrix represents a link that must be removed to create a dominance graph, i.e., $y_{ij} = 1$. Thus, there is a one-to-one correspondence between each member of P and a binary solution to the rankability LP. STEP 4: Assume there exists one member of P, called P_1, that is not an extreme point. This implies P_1 can be written as a convex combination of at least two other extreme points. A convex combination of binary extreme points must be fractional. Yet this contradicts the fact that all members of P are binary. Thus, P_1 must be an extreme point. ∎

THEOREM 5.2. *The centroid of the optimal face of the rankability IP is a convex*

combination of all optimal solutions, and, thus, all members of the set P. Fractional elements of this centroid solution represent disagreements, according to the members of P, between the rank positions of items.

Proof. By Theorem 5.1, this centroid of the optimal face must be a convex combination of the $|P|$ binary extreme points on the optimal face. It remains to show that fractional elements of the centroid represent disagreements, according to the members of P, between the rank positions of items. We do so by contradiction. Assume fractional element x_{ij} (w.l.o.g.) of the optimal solution represents agreement, according to the members of P, between the rank positions of items i and j. Assume, w.l.o.g., that $i > j$. By Theorem 5.1, each member of P corresponds to a binary optimal extreme point of the LP, so that $\mathbf{X}$ is binary. In particular, the (i, j)-element is binary for every member of P and specifically $x_{ij} = 1$ since $i > j$. Thus, for the (i, j) position, every convex combination $\alpha_1, \alpha_2, \ldots, \alpha_P$ with $\alpha_1 + \alpha_2 + \ldots + \alpha_P = 1$ of the members of P has $\alpha_1(1) + \alpha_2(1) + \ldots + \alpha_P(1) = 1$. Yet this contradicts the fact that x_{ij} is fractional. Thus, a fractional x_{ij} represents disagreements among the members of P about the rank positions of i and j. ∎

Since the LP interior point solution approximates the centroid of the IP's optimal face, then Theorem 5.2 approximately holds for the LP solution and this is good news. Finding the full set P amounts to finding all optimal solutions of the rankability problem, which is known to be very expensive for general IPs and LPs. Instead, without actually forming P, the matrices $\mathbf{P}_a$, $\mathbf{P}_d$, $\mathbf{P}_>$ used to make suggestions on improving the graph's rankability can be approximated with the LP centroid solution. Figure 9 demonstrates this point with an $n = 6$ example graph. The first row of Figure

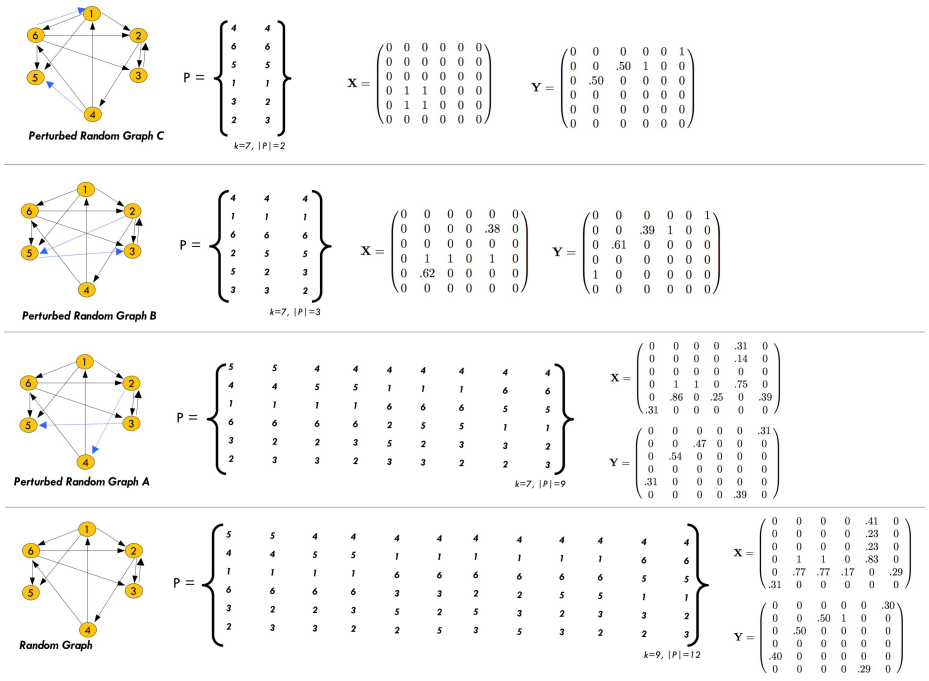

FIG. 9. ***LP outputs matrices $\mathbf{X}$ and $\mathbf{Y}$.*** *The location of fractional elements in these matrices gives information about the set P without the expense of actually forming the full P, thus creating great computational savings.*

9 pertains to the graph denoted Random Graph C. The LP outputs a $\mathbf{Y}$ matrix with fractional values in positions $(2,3)$ and $(3,2)$. Compare this with the P set, which shows that there is maximal indecision about how to rank items 2 and 3. Half of the members of P have $2 > 3$ and half have $3 > 2$. For Random Graph B, the second row of Figure 9, the LP outputs $\mathbf{X}$ and $\mathbf{Y}$ matrices with fractional values in positions $(2,3)$ and $(3,2)$ and $(2,5)$ and $(5,2)$. Again, notice that these fractional values approximate the proportion of indecision in the set P between these pairs of items. For example, $\mathbf{X}(5,2) = .62$ and this approximates the information in P, which shows that two-thirds of its rankings have $5 > 2$. This pattern continues for the two other graphs, Random A and Random. Recall that the IP's $\mathbf{X}$ matrix gives binary information about which links to add to improve rankability. *The LP relaxation's* $\mathbf{X}$ *matrix then gives a* ***probabilistic*** *interpretation about which links to add.* See the supplement for an $n = 12$ example that demonstrates this statement with real data from the 2009 ACC college basketball season.

From the four examples of Figure 9, the ACC example in the supplement, and Theorem 5.2, we make several observations:

1. $\mathbf{X} \approx \mathbf{P}_a$. Recall that $\mathbf{P}_a(i,j)$ gives the percentage of rankings in P that suggest adding a link from i to j.
2. $\mathbf{Y} \approx \mathbf{P}_d$. Recall that $\mathbf{P}_d(i,j)$ gives the percentage of rankings in P that suggest deleting a link from i to j.
3. $\mathbf{D} + \mathbf{X} - \mathbf{Y} \approx \mathbf{P}_>$. Recall that $\mathbf{P}_>(i,j)$ gives the percentage of rankings in P that rank i above j.

Summary: The matrices $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{D} + \mathbf{X} - \mathbf{Y}$ from the LP solution approximate the matrices $\mathbf{P}_a$, $\mathbf{P}_d$, $\mathbf{P}_>$ that are built from the set P. The LP solution does not form P, yet is still able to create approximations to these valuable matrices.

6. Experiments. The purpose of this section is to: (1) work with real data to demonstrate the type of findings that rankability enables and (2) assess the accuracy and runtimes of the approximate methods of Section 5.2.

6.1. Real data. In this section, we experiment with two datasets: (1) an NFL dataset of the $n = 8$ AFC East and AFC West football teams and (2) a Movielens collection of $n = 100$ movies. There are several ways to transform real data to prepare it for rankability analysis. The dominance matrix ($\mathbf{D}$ for unweighted graphs and $\mathbf{C}$ for weighted graphs) is the sole input to the rankability problem. For some applications, the creation of this matrix is natural. For example, in sports, dominance relations are straightforward: an unweighted matrix $\mathbf{D}$ matrix uses $\mathbf{D}_{ij} = 1$ if team i beat team j whereas a weighted $\mathbf{C}$ matrix might use point scores if team i beat team j by a score of 52 to 49 so that $\mathbf{C}_{ij} = 52$ and $\mathbf{C}_{ij} = 49$. There is some art to the modeling here. For instance, another definition of $\mathbf{C}$ uses the point differential so that $\mathbf{C}_{ij} = 3$ and $\mathbf{C}_{ij} = 0$ for the 52-49 win of i over j. For the NFL examples of this section, we created a weighted $\mathbf{C}$ matrix where $\mathbf{C}_{ij}$ is the number of times team i beat team j.

Other applications require a bit more ingenuity in the definition of dominance. The Movielens dataset consists of user ratings (on an integer scale of 1 to 5) of movies, which creates a rectangular movie-by-user matrix. To create a square movie-by-movie weighted dominance matrix $\mathbf{C}$ we defined $\mathbf{C}_{ij}$ as the percentage of users rating both movie i and movie j who rated $i > j$. Once again, there are other options and the art of modeling incorporates any domain knowledge associated with the application. For instance, should ties (when a user rated both movies i and j and rated them equally) be considered or omitted? How this question is answered may affect the results and thus should be answered by the domain expert.

Figure 10 summarizes the rankability analysis for the NFL and movie datasets. The left side of the figure shows weekly rankability results for $n = 8$ teams during the 2016 NFL season. The right side of the figure shows results for the $n = 100$ movie dataset. The graph at the top plots both the original rankability measure r in the solid line and a transformed measure r_t in the dashed line over the 17 weeks of the seasons. As noted earlier and these and other experiments confirm, r tends to push scores near 1, whereas a new transformed measure $r_t = \frac{k}{k_{max}p(1-\tau)}$ is designed to provide better differentiation in rankability scores. This transformed r_t measure starts with the distance k from the ideal as a percentage then penalizes the score for the size and diversity (as measured by the average Kendall τ) of the set P. Zooming in on the solid line, shows that the two measures track the same trends. By Week 6, the rankability scores have settled down, meaning Week 6 rankings have more validity than Week 1 rankings. The $\mathbf{R}$ matrix shows the confidence in the rankings in a visual form. Recall that $\mathbf{R}_{ij}$ is the number of rankings in P that have team i in rank position j. Notice from Week 1 to 6 to 17 how the $\mathbf{R}$ matrix changes. At Week 1, there is little information and most of the teams range all over in rank. There is enough information to believe that Kansas City belongs toward the top of the ranking and the LA Chargers toward the bottom. By Week 6, Kansas City emerges as the strong leader while New England is one of the teams toward the bottom. Then at the end of the season, Week 17, New England and Kansas City dominate the top rank positions and it is clear that Buffalo belongs at the bottom.

The next row of Figure 10 provides information about the nature of the available data and is derived from the $\mathbf{P}_a$ and $\mathbf{P}_d$ matrices. In Weeks 1 and 6, a quick visual comparison shows that there are many more nonzeroes in $\mathbf{P}_a$ than $\mathbf{P}_d$, meaning that many more matchups are needed to improve the rankability of the data. By Week 17, $\mathbf{P}_a$ has fewer nonzeroes and $\mathbf{P}_d$ now has some nonzeroes, meaning that some of the data are now inconsistent with the rankings. The matrix $\mathbf{P}_d$ also provides information about potential noise. In this example, it identifies Buffalo's win over New England as a fluke win. In the 2016 season, New England started slow but then went on to its usual dominance and the lowly Buffalo beating mighty New England was considered a fluke by sports analysts.

The final row of Figure 10 provides suggestions about which matchups could improve the rankability of the data. Of course, in the NFL this is impractical during the season but could have applications in scheduling the next season. However, for the movie application, this information is very practical. Companies such as Netflix or Amazon have the ability to solicit pairwise dominance relations with the aim of improving their overall global ranking of items. The visual plots are created by forming an upper triangular matrix $\mathbf{I}_{ij} = |(\mathbf{D} + \mathbf{P}_a - \mathbf{P}_d)_{ij} - (\mathbf{D} + \mathbf{P}_a - \mathbf{P}_d)_{ji}|$. Elements in $\mathbf{I}$ that are near 0 mean that there is great indecision as to how to rank the two items and these are the items that are listed as most helpful in improving rankability.

The final column of Figure 10 presents results for the movie data and provides a comparison between exact and approximate results. For the NFL data, we used the exact method of Section 5.1 to produce the full set P and matrices derived from that set such as $\mathbf{R}$, $\mathbf{P}_a$, $\mathbf{P}_d$, and $\mathbf{I}$. For comparison, for the movie data, we used an approximate method, the LP Interior Point Solution of Section 5.2, which does not produce the full set P and instead approximates some of the resulting matrices. Recall that the LP's $\mathbf{X}$ and $\mathbf{Y}$ matrices approximate $\mathbf{P}_a$ and $\mathbf{P}_d$. As a result, we can create an approximation to $\mathbf{I}$, and thus, present suggestions on which links to add or remove in order to improve rankability. The final column of Figure 10 shows

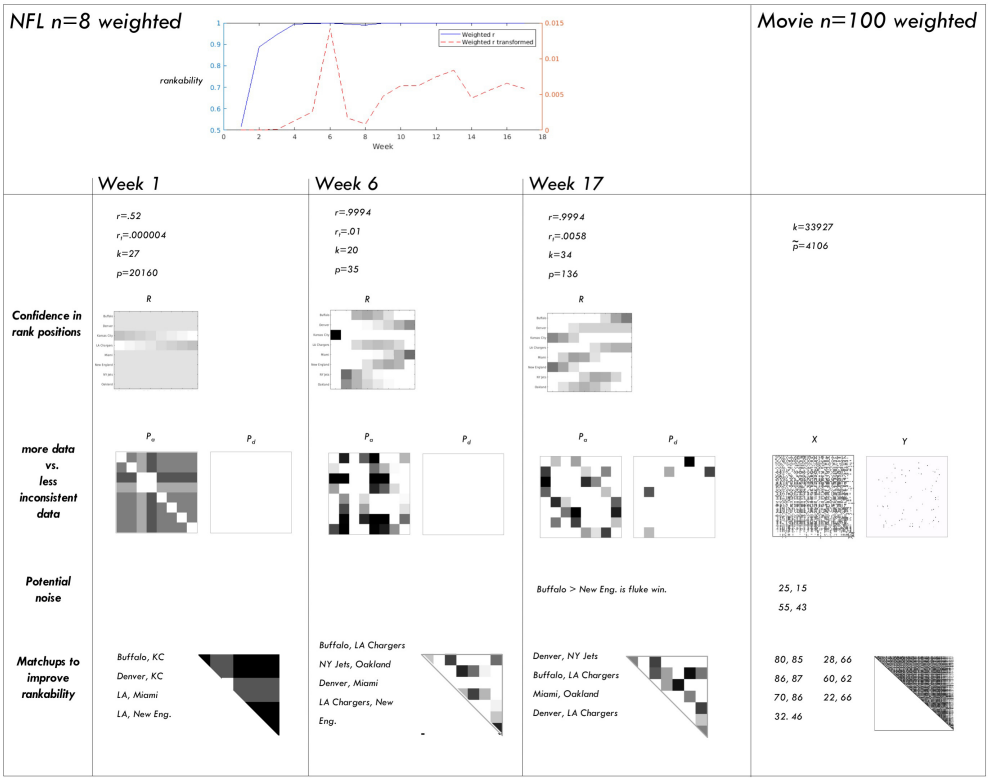

FIG. 10. ***Infographic summarizing rankability on real data.*** *The data on the left is from the $n = 8$ teams in the 2016 NFL AFC East and West divisions. The data on the right is from $n = 100$ movies from the Movielens dataset.*

clearly what is lost with this particular approximate method: namely, the value of p, and hence, the rankability measure r, and the $\mathbf{R}$ matrix that visually displays the confidence of items in rank positions. However, we argue that these losses are not so important when compared to what is recovered. For most applications, knowing the precise value for a rankability score r, e.g., that a graph's rankability is, say, $r = .98$, is not as useful as knowing that gathering more information about specific pairs of items will improve rankability.

6.2. Runtimes and Accuracy. The table below uses an artificial $n = 20$ dataset to study the runtimes and accuracy of the approximate rankability methods. This dataset that was generated by removing 75% of the links at random in a dominance graph. Table 3 shows the runtimes of the exact method and the approximate methods (with serial implementations). The accuracy of the approximate methods is assessed with the four rightmost columns. These experiments were run on a single node with 2 CPUs, 24 cores, and 128 GB of memory.

Approximate Method 1 is reported in rows 2-5. For example, "PoolSearch 110; 5 reorderings" means that Gurobi's PoolSearch routine is set to run until 100 multiple optimal solutions (i.e., members of P) are produced and then restarted 5 times each with a different random reordering of the $\mathbf{D}$ matrix. Though increasing the number of reordering restarts in Approximate Method 1 increases the processing time in a serial implementation, it produces more members of P, and hence better approximations

TABLE 3
Runtimes and Accuracy of Approximate Methods

	time	$\frac{\lvert\hat{P}\rvert}{\lvert P\rvert}$	$\frac{\lVert P_a-\hat{P}_a\rVert}{\lVert P_a\rVert}$	$\frac{\lVert P_d-\hat{P}_d\rVert}{\lVert P_d\rVert}$	$\frac{\lVert P_>-\hat{P}_>\rVert}{\lVert P_>\rVert}$
LP interior point solution	11s	n/a	10.2	7.5	3.4
PoolSearch 100; 5 reorderings	14s	.14	12.5	8.3	3.8
PoolSearch 100; 10 reorderings	26s	.28	6.1	5.4	1.9
PoolSearch 100; 15 reorderings	39s	.37	3.7	3.0	1.1
PoolSearch 100; 20 reorderings	51s	.43	5.2	2.8	1.6
Exact Method with Pruning	4m7s	1	1	1	1

to the $\mathbf{P}_a$, $\mathbf{P}_d$, $\mathbf{P}_>$ matrices that are used to make suggestions about improving rankability. Thus, a user can adjust the runtime of Approximate Method 1 to control the size of the set P. In addition, Approximate Method 1 is easily parallelizable; each restart can be sent to its own processor.

The other approximate method, the LP Interior Point Solution, seems promising. It is the least expensive approximate method in a serial implementation and though it does not create a partial set $\hat{P}$, it still gives a great deal of rankability information. The supplement provides more accuracy results for this method.

Graphs of size $O(10^1) \leq n \leq O(10^2)$, such as those used for Tables 1 and 3, are often adequate for many practical applications. Consider companies such as Amazon and Netflix, where products are often subdivided into finer and finer groups. Netflix is interested in improving the rankability of its top 50 movies in the Animated–Japanime genre. Amazon similarly wants to understand the rankability of the 100 products in the category of Kids–Toys–Indoor–Preschool–Books.

7. Conclusions. This paper posed and solved the new *rankability problem*, which refers to a dataset's inherent ability to produce a meaningful ranking of its items. We created a measure r that determines a graph's rankability by accounting for two factors: (1) how far the graph is from a perfect dominance graph–in particular how many links k must be added or removed to transform the graph to the perfect dominance graph, and (2) how many rankings p can be produced with this number of changes to the graph. A near-perfect rankability score near 1 means that there is little indecision about how to rank the items and, in practice, it is likely that every ranking method will produce the same ordering. A poor rankability score near 0 means that the data is both far from the perfect ranking graph and there are many rankings equally far from the perfect ranking graph and, in practice, ranking methods will likely produce very different rankings.

We posed the rankability problem as a combinatorial optimization problem that we solved with several exact and approximate algorithms with both serial and parallel implementations. Applying our new measure and algorithms to both artificial and real datasets revealed its ability to not only assign an algorithm-agnostic rankability score to a dataset but to also make suggestions on how to improve the dataset's rankability.

8. Future Work. Our definition for rankability imposes a strict linear ordering that does not allow for ties. This restriction may be limiting for some applications. For instance, perhaps Netflix would be more interested in ranking movies into equivalence classes. Our experiments show that the set P contains much equivalence information and, as future work, we plan to create algorithms and visualizations that display such information.

This paper described our combinatorial approach to a rankability measure. We are also exploring a linear algebraic approach to rankability built from the singular value decomposition. Finally, yet another direction for future work pursues the identification of subsets within the data that are highly rankable and subsets that are highly unrankable.

Acknowledgments. We thank the editor-in-chief and anonymous referees for their suggestions, which greatly improved the paper.

REFERENCES

[1] Andreas Adolfsson, Margareta Ackerman, and Naomi C. Brownstein. To cluster, or not to cluster: How to answer the question. In *Proceedings of Knowledge Discovery from Data, Halifax, Nova Scotia, Canada*, pages 1–9, 2017.
[2] Iqbal Ali, Wade D. Cook, and Moshe Kress. On the minimum violations ranking of a tournament. *Management Science*, 32(6):660–672, 1986.
[3] Ronald D. Armstrong, Wade D. Cook, and Lawrence M. Seiford. Priority ranking and consensus formation: The case of ties. *Management Science*, 28(6):638–645, 1982.
[4] C. Richard Cassady, Lisa M. Maillart, and Sinan Salman. Ranking sports teams: A customizable quadratic assignment approach. *INFORMS: Interfaces*, 35(6):497–510, 2005.
[5] Timothy P. Chartier, Erich Kreutzer, Amy N. Langville, and Kathryn E. Pedings. Sensitivity of ranking vectors. *SIAM Journal on Scientific Computing*, 33(3):1077–1102, 2011.
[6] Timothy P. Chartier, Amy N. Langville, Kevin Hutson, and Michael W. Berry. Identifying influential edges in a directed network: big events, upsets, and non-transitivity. *Complex Networks*, 2013.
[7] B. Jay Coleman. Minimizing game score violations in college football rankings. *INFORMS: Interfaces*, 35(6):483–496, 2005.
[8] Wade D. Cook and Lawrence M. Seiford. Priority ranking and consensus formation. *Management Science*, 24(16):1721–1732, 1978.
[9] Komei Fukuda and Alain Prodon. *Double description method revisited*, pages 91–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.
[10] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
[11] Martin Grotschel, Michael Junger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984.
[12] Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial Hodge theory. *Mathematical Programming*, 127(1):203–244, 2011.
[13] Maurice G. Kendall and B. Babington Smith. On the method of paired comparisons. *Biometrica*, 31, 1939.
[14] Amy N. Langville and Carl D. Meyer. *Who's #1:The Science of Rating and Ranking Items*. Princeton University Press, Princeton, 2012.
[15] Amy N. Langville, Kathryn Pedings, and Yoshitsugu Yamamoto. A minimum violations ranking method. *Optimization and Engineering*, pages 1–22, 2011.
[16] Rafael Marti and Gerhard Reinelt. *The Linear Ordering Problem: exact and heuristic methods in combinatorial optimization*. AMS, 2011.
[17] Sanjay Mehrotra and Yinyu Ye. Finding an interior point in the optimal face of linear programs. 62:497–515, 02 1993.
[18] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, Philadelphia, 2000.
[19] Jarrod Moore and Margareta Ackerman. Foundations of perturbation robust clustering. In *Proceedings of 2016 IEEE 16th International Conference on Data Mining*, 2016.
[20] T.S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*. Princeton University Press, 1953.
[21] Alantha Newman and Santosh Vempala. *Fences Are Futile: On Relaxations for the Linear Ordering Problem*, pages 333–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
[22] M. Oswald, G. Reinelt, and H. Seitz. Applying mod-k-cuts for solving linear ordering problems. *TOP*, 17(1):158–170, Jul 2009.
[23] Juyong Park. On minimum violations ranking in paired comparisons, Nov 2005.
[24] Gerhard Reinelt. *The Linear Ordering Problem: Algorithms and Applications*. Heldermann Verlag, 1985.

765 [25] Gerhard Reinelt, M. Grötschel, and M. Jünger. Facets of the linear ordering polytope. *Mathe-*
766 *matical Programming*, 33:43–60, 1985.
767 [26] Stan Schenkerman. Inducement of Nonexistent Order by the Analytic Hierarchy Process. *De-*
768 *cision Sciences*, 28(2):475–482, 1997.