Algorithms for Discovering Bucket Orders from Data

Aristides Gionis
HIIT Basic Research Unit
University of Helsinki

Heikki Mannila
HIIT Basic Research Unit
Helsinki University of Technology and
University of Helsinki

Kai Puolamäki
HIIT Basic Research Unit
Helsinki University of Technology

Antti Ukkonen
HIIT Basic Research Unit
Helsinki University of Technology

ABSTRACT

Ordering and ranking items of different types are important tasks in various applications, such as query processing and scientific data mining. A total order for the items can be misleading, since there are groups of items that have practically equal ranks.

We consider bucket orders, i.e., total orders with ties. They can be used to capture the essential order information without overfitting the data: they form a useful concept class between total orders and arbitrary partial orders. We address the question of finding a bucket order for a set of items, given pairwise precedence information between the items. We also discuss methods for computing the pairwise precedence data.

We describe simple and efficient algorithms for finding good bucket orders. Several of the algorithms have a provable approximation guarantee, and they scale well to large datasets. We provide experimental results on artificial and a real data that show the usefulness of bucket orders and demonstrate the accuracy and efficiency of the algorithms.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications - data mining

General Terms: Algorithms, Performance, Theory

Keywords: Ordering, ranking, partial order, bucket order

1. INTRODUCTION

Ordering and ranking items of different types are important tasks in various applications, such as ranking database query results [1, 6, 11, 12, 13, 19, 22], web ranking [4, 5, 10, 17, 20], finding orders of items in data mining [16, 23, 27], and in machine learning [7, 8, 15, 21]. The requirements for the results of ordering and ranking vary according to the application. In ranking answers to a web search query it is important to get the most relevant pages among the top 10 (or so) of the answers, so that the user can select the relevant link quickly; the relative ordering of irrelevant pages is not important. In other applications, such as processing scientific queries or aggregating preference information, the ordering of the whole set of answers is of interest.

Consider, as an example, the question of aggregating movie preference data. Given a set of movies, viewers have given grades to the some of the movies. The grading scale typically has low granularity: there might be 5–10 different grades. The rankings contradict each other frequently, and the scales used by different persons can be very different, so it is well motivated to ask what would be a sound and efficient way of combining the rankings of the individual viewers.

Finding a total order for all the movies would be an overfit to the data: the distinction between the movies ranked 562nd and 563rd is probably not significant. It is much more meaningful to find a coarser classification of the movies into groups and the relative ordering of those groups (corresponding to movies that can be labeled from masterpieces or excellent to really bad or pathetic).

In this paper, we consider the discovery of *bucket orders* from data. A bucket order [11, 28] (also called a weak order) is a total order with ties. More formally, a bucket order $B = \langle M_1, \ldots, M_k \rangle$ on the set of items M is a partial order on M defined by a partition of M into k buckets, $M_1, \ldots, M_k$. The items within a bucket are incomparable, while all elements in bucket M_i precede all elements in bucket M_j, if $i < j$. Note that, in the previous example, a ranking of movies by a viewer into classes 1–5 is a bucket order. In general, a bucket order B on the set M can also be viewed as a matrix C^B: the entry C^B_{tu} is $\frac{1}{2}$ if t and u belong to the same bucket, $C^B_{tu} = 1$ if t precedes u in the bucket order, and $C^B_{tu} = 0$ if t follows u in the bucket order.

Our approach is as follows. Given a set of rankings on subsets of the set M of items, we compute the *pair order matrix* C. The entry C_{tu} indicates how many users preferred item t to item u. The pair order matrix is normalized so that $C_{tu} + C_{ut} = 1$ for all t and u. Then we search for a bucket order B, such that C^B approximates C in the best way possible as determined by an L_1 norm between C and C^B.

Given a pair order matrix, the problem of finding the bucket order that minimizes the L_1 norm is NP-hard. Hence one needs to look for approximation algorithms. Ailon et al. [2] describe an insightful randomized algorithm for the feedback arc set problem, i.e., for finding *total* orders. We adapt the *pivot algorithm* of [2] to finding bucket orders and analyze its approximation behavior. We prove that the al-

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

gorithm achieves an approximation ratio of 9; if the entries of the pair order matrix satisfy also the triangle inequality, then the approximation ratio is 5. We also show that the expected running time of the algorithm on any input is $O(n \log n)$ for n items. This means, incidentally, that the algorithm does not inspect all the entries of the pair order matrix.

Bucket orders can also be discovered by grouping items into buckets from a total order. We briefly describe two algorithms based on this approach and compare their performance to the pivot algorithm in the empirical section.

The rest of this paper is organized as follows. In Section 2 we describe the basic bucket order model. Section 3 discusses the properties of bucket orders and gives some simple results. The pivot algorithm is given in Section 4, where we also give some theoretical results related to its approximation ratio and expected running time. Alternative approaches to finding bucket orders are given in Section 5, while Section 6 presents our experimental results. Related work is considered in Section 7, and finally in Section 8 we give our concluding remarks and we discuss possible extensions of the model.

2. BASIC DEFINITIONS

A partial order on a set of items M is a reflexive, antisymmetric, and transitive binary relation on M. A total order T is a partial order such that for all pairs of items $u, t \in M$ either $(u,t) \in T$ or $(t,u) \in T$. For many applications, modeling the data with a total order might be a too strict. On the other hand, arbitrary partial orders can be too complex for user interface requirements.

We focus on the study of bucket orders, a class of partial orders between total orders and arbitrary partial orders. A bucket order B for a set of items M is defined by a sequence of k buckets, $\langle M_1, \ldots, M_k \rangle$, where the nonempty sets $\{M_j\}_{j \in \{1,\ldots,k\}}$ form a partition of M. The items within a bucket, $u, t \in M_j$ are unordered, i.e., $(u,t) \notin B$ and $(t,u) \notin B$, whereas the buckets are totally ordered, i.e., for every $u \in M_i$ and $t \in M_j$, $i < j$, we have $(u,t) \in B$.

In the applications we consider in this paper we assume that we can obtain *pairwise* ordering information between items of M. That is, we assume that we have an $|M| \times |M|$ *pair order matrix* C, where entry C_{tu} measures the *evidence* or *probability* that the item t precedes the item u. We assume that the values C_{tu} are in the interval $[0,1]$. Furthermore we assume that for all t, u we have

$$C_{tu} + C_{ut} = 1, \quad C_{tu} \geq 0 \quad \text{(probability constraint)}. \tag{1}$$

In many cases it is also reasonable to assume that the entries of C matrix satisfy the triangle inequality, that is, for all t, u, and v we have

$$C_{tu} \leq C_{tv} + C_{vu} \quad \text{(triangle inequality)}. \tag{2}$$

Any partial order P can be represented with a pair order matrix C^P satisfying the probability constraint and the triangle inequality, the former capturing antisymmetry, and the latter capturing transitivity. The entries of C^P are

$$C^P_{tu} = \begin{cases} 1 & (t,u) \in P \\ 0 & (u,t) \in P \\ \frac{1}{2} & (t,u) \notin P \text{ and } (u,t) \notin P, \end{cases}$$

for all $t \neq u$, and $C^P_{tt} = \frac{1}{2}$.

Total orders and bucket orders are special cases of partial orders, and hence they can be represented as pair order matrices. The pair order matrix C^T for a total order T can be rearranged so that the all diagonal entries are equal to $\frac{1}{2}$, the entries above the diagonal are equal to 1, and the entries below the diagonal are equal to 0. Similarly, the pair order matrix C^B for a bucket order B can be rearranged to a *block diagonal* matrix, so that the entries below, inside, and above the diagonal blocks are equal to 0, $\frac{1}{2}$, and 1, respectively.

Given a dataset of items and a pair order matrix on the items we want to find the best bucket order describing this data. As a cost function for measuring the quality of the solution we use distance between matrices. If C and D are two matrices, we define

$$L(C,D) = \sum_{t \neq u} g(C_{tu}, D_{tu}),$$

where g is a symmetric non-negative function satisfying $g(x,x) = 0$, and in particular

$$L_1(C,D) = \sum_{t \neq u} |C_{tu} - D_{tu}|. \tag{3}$$

Other distance measures between matrices are also possible, but not considered in this paper.[1] We now define the *bucket order problem*.

PROBLEM 1. *Let C be a pair order matrix on a set of items M. Find a bucket order B on the items of M such that the distance measure $L_1(C, C^B)$ is minimized.*

In other words, the task is to find a bucket order B on the items of M, minimizing the sum of values $|C_{tu} - 1|$ if $(t,u) \in B$, $|C_{tu}|$ if $(u,t) \in B$, and $|C_{tu} - \frac{1}{2}|$ if $(t,u) \notin B$ and $(u,t) \notin B$, for all pairs (t,u).

Example. For a very simple example to motivate our problem formulation and also contrast with previous work by Fagin et al. [13], consider the following dataset. We have only two items $M = \{a, b\}$ and a set of 100 sequences, permutations of M, such that 51 of them rank a before b, and the rest 49 rank b before a. Then the pair order matrix will be

$$C = \begin{bmatrix} 0.5 & 0.51 \\ 0.49 & 0.5 \end{bmatrix},$$

and the optimal bucket order according to Problem 1 will place a and b in the same bucket, which is a very intuitive solution. On the other hand, the *median-aggregation algorithm* proposed in [13] will select a final ranking that will place a before b. This example demonstrates that the two ranking models, the one considered in this paper and the one in [13], have different objectives and different properties.

Obtaining the pair order matrix. We construct the pair order matrix C by using a set of total orders of the items in M. This is done by setting C_{tu} to be the fraction of the total orders in which t precedes u. If each of the total orders ranks all items in M, the resulting pair order matrix will satisfy both the probability and triangle inequality constraints. If the total orders rank only proper subsets of M,

[1] Notice that denoting the distance measure in Equation (3) by L_1 is a slight abuse of notation, since the L_1 matrix norm is usually meant to be the maximum over all matrix columns of the sum of the absolute values of the entries in the column.

the resulting pair order matrix is guaranteed to satisfy only the probability constraint.

3. PROPERTIES OF THE MODEL

In this section we discuss basic properties of the bucket order problem.

Number of bucket orders. We first note that given n items, the number of distinct bucket orders with k nonempty buckets is given by the Stirling numbers of the second kind $S(n,k)$. The total number $S(n)$ of distinct bucket orders on n items satisfies

$$S(n) \sim \frac{n!}{2(\log 2)^{n+1}},$$

for example, see sequence number A000670 in [26], and [28].

NP-hardness. We show that the bucket order problem is NP-hard. The proof idea is based on the concept of *tournaments*, which are *complete* directed graphs with no two-node cycles. A pair order matrix C with 0–1 values, i.e., $C_{tu} = 1$ or $C_{tu} = 0$, whenever $t \neq u$, corresponds to a tournament graph. We first have the following.

OBSERVATION 1. *Let G be a tournament graph. Then the cost of the optimal total order for G is equal to the cost of the optimal bucket order.*

PROOF. Let $T^*(G)$ and $B^*(G)$ be the costs of the optimal total order and optimal bucket order for G, respectively. We show that $T^*(G) = B^*(G)$.

First it is clear that $B^*(G) \leq T^*(G)$, since any total order is also a bucket order. On the other hand, we have $T^*(G) \leq B^*(G)$. Assume that the bucket order that achieves $B^*(G)$ is not a total order. We convert that optimal bucket order to a total order by ordering the vertices within buckets one bucket at a time. For each bucket, any ordering within the bucket does not change the costs with respect to the vertices outside the bucket, only the vertices inside the bucket have influence on the cost. Since G is a tournament, for a bucket with k vertices, the cost inside the bucket is $k(k-1)/2$. Take any random permutation π of the vertices inside the bucket, and consider also the reverse permutation π' that orders the vertices in the reverse order than π. There are a total of $k(k-1)$ edges, and each edge is charged to either π or π', so one of the two permutations should have cost no more than $k(k-1)/2$. By applying the same argument and linearizing the items in each bucket, we can create a total order that has cost at most $B^*(G)$. □

THEOREM 1. *Given a pair order matrix C, the problem of finding the optimal bucket order for C is NP-hard.*

PROOF. From Observation 1, for pair order matrices corresponding to tournament graphs, finding the optimal bucket order is equivalent to finding the optimal total order. Since the latter problem is NP-hard [3], the bucket order problem is also NP-hard. □

At this point, it is natural to ask if the above argument can be generalized and, in fact, prove that there is no additional cost improvement in finding a bucket order over finding the optimal total order. However, this is not the case; if the pair order matrix has a clear bucket structure then the cost of the optimal bucket order can be significantly better than the cost of the optimal total order.

A single parameter. Our problem formulation does not require providing the number of buckets: bucket orders with unnecessarily small or unnecessarily large number of buckets are penalized automatically from the objective function. Thus, finding the optimal bucket order using the formulation of Problem 1 is a parameter-free task.

Note that that our main algorithm uses one parameter β whose value influences the final number of buckets, however, (i) the effect is not so dramatic, (ii) $\beta = \frac{1}{4}$ is a default value that can be used in all cases, and which provides the approximation guarantee of the algorithm.

4. THE PIVOT ALGORITHM

4.1 Description of the algorithm

The *bucket pivot algorithm* is a sorting algorithm that creates a bucket order. The algorithm selects a random pivot element, after which it compares the pivot with the other items and divides them to three classes: "left", "same" and "right". The algorithm then recurses to bucket sort the "left" and "right" classes. The "same" class contains, in addition to the pivot, all items for which there is no clear relative ordering with respect to the pivot. The "same" class will be output as a bucket.

The bucket pivot algorithm for a set of items M and a pair order matrix C, BP(M,C,β), is given as pseudocode in Algorithm 1. The algorithm outputs a bucket order. The algorithm has one parameter, β which defines the limit by which the items are put to the same bucket. Unless otherwise mentioned, we use $\beta = \frac{1}{4}$. For $\beta = 0$, the bucket pivot algorithm returns a total order and is equivalent to the FAS-PIVOT algorithm of [2], and if $\beta > \frac{1}{2}$ the algorithm always returns only one bucket.

4.2 Theoretical results

Our results are summarized in the following theorems. The first one states that the pivot algorithm has a bounded approximation ratio.

THEOREM 2. *Let C be a pair order matrix, BP the bucket order found by the pivot algorithm with $\beta = \frac{1}{4}$, and OPT be the optimal bucket order for matrix C. Furthermore, let L^{BP} be the expected cost of the solution found by the pivot algorithm, that is, $L^{BP} = E\left[L_1(C^{BP}, C)\right]$, and L^{OPT} be the cost of the optimal solution. In the general case we have $L^{BP} \leq 9L^{OPT}$. If C satisfies also the triangle inequality we have $L^{BP} \leq 5L^{OPT}$. For restricted matrices with values in $\{0, \frac{1}{2}, 1\}$, the bounds without and with triangle inequality, are $L^{BP} \leq 5L^{OPT}$ and $L^{BP} \leq 3L^{OPT}$, respectively.*

The proof follows the ideas presented in [2]. Details will be available in the full version of this paper.

It is not hard to see that an input matrix corresponding to a total order results in the worst-case running time of $O(n^2)$. However, for the expected number of comparisons we can state the following.

THEOREM 3. *The expected number of comparison operations made by the bucket pivot algorithm is $O(n \log n)$.*

The proof is similar to the one used to show the expected running time of randomized quicksort. Since the size of the

Algorithm 1 The Bucket Pivot Algorithm. By default, we use $\beta = \frac{1}{4}$.

BP(V,C,β) {Input: V, set of items; C, pair order matrix; $\beta \geq 0$, parameter. Output: Bucket order.}
if $V = \emptyset$ **then**
  return $\emptyset$
end if
Pick a pivot $t \in V$ uniformly at random.
$L \leftarrow \emptyset$
$S \leftarrow \{t\}$
$R \leftarrow \emptyset$
for all items $u \in V \setminus \{t\}$ **do**
  if $C_{tu} < \frac{1}{2} - \beta$ **then**
    Add u to L.
  else if $\frac{1}{2} - \beta \leq C_{tu} < \frac{1}{2} + \beta$ **then**
    Add u to S.
  else if $\frac{1}{2} + \beta \leq C_{tu}$ **then**
    Add u to R.
  end if
end for
return order $\langle \mathrm{BP}(L, C, \beta), S, \mathrm{BP}(R, C, \beta)) \rangle$

input matrix C is $O(n^2)$, but the expected running time is only $O(n \log n)$, the algorithm in general does not inspect all elements of C. In some cases it is thus possible to compute only those elements of C that are needed.

5. ALTERNATIVE ALGORITHMS

Both of the alternative algorithms are based on the following idea: Given a total order of the items we can sort the columns of the pair order matrix according to it. A bucket order can then be constructed by viewing the columns of the pair order matrix as an n-dimensional time-series which is segmented to k segments. Segmentation can be done optimally by using dynamic programming, so the only question is how to select the total order.

The first alternative is to take any total order T that is a linear extension of the bucket order given by the bucket pivot algorithm. Another approach is to consider the rowsums of the matrix C. The ith rowsum is simply the sum of all elements on row i. The total order can be formed by ordering the items in increasing (or decreasing) order of the rowsums.

Running time of segmentation by dynamic programming is quadratic in n. This can be overcome by using some alternative segmentation algorithm. We use *globally iterative replacement* (GIR), proposed in [18]. For time-series data, GIR tends to produce segmentations very close to the optimal ones given by the dynamic programming algorithm.

The two alternative algorithms are called BP-GIR and RS-GIR, depending if the pivot algorithm (BP) or rowsums (RS) are used to determine the initial total order.

6. APPLICATIONS AND EXPERIMENTS

We used both artificial and real datasets for studying the algorithms' performance. In the following B denotes a bucket order and C^B is the corresponding pair order matrix. In all cases the parameter β used in the pivot algorithm was 0.25.

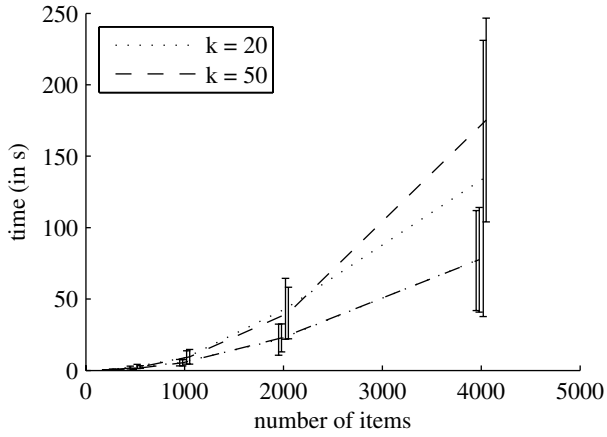

Figure 1: Median (of 50 runs) running times of the pivot algorithm (BP) with different number of items. k denotes the number of buckets in the input. Rows of the pair matrix were computed on the fly based on a set of either 1000 or 2000 artificially generated sequences (total orders). The two upper curves show the running times for 2000 sequences. At 4000 items k has a small effect on the running time. In case of 1000 sequences (lower curves) k has no effect.

6.1 Artificial data

We use artificial data to demonstrate the algorithms' scalability and noise tolerance. Scalability of BP-GIR and RS-GIR was not tested, because segmenting a time series of both length and dimension n is not feasible for very large n, even when global iterative replacement is used.

Synthetic input data is generated with the following procedure. First we pick uniformly at random a bucket order G on n items with k buckets. Then we construct a set S of sequences of the n items, such that a fraction ρ of the sequences are completely random and the rest are linear extensions of G. The pair order matrix C is obtained by counting how many times item i precedes item j in the sequences and normalizing the resulting matrix such that the probability constraint holds.

To test the scalability we created datasets containing 1000 or 2000 sequences with different values of n ($n = \{250, 500, 1000, 2000, 4000\}$) and k ($k = \{10, 20, 50\}$). The level of noise ρ was fixed to 0.2. We ran the BP algorithm 50 times for each input (using the same input each time) and measured the running time on each round. Results are shown in Figure 1.

For testing the noise tolerance a MATLAB implementation was used. Note that in general the pivot algorithm does not determine the complete error $L_1(C^G, C^B)$, since this would require $O(n^2)$ steps. The test was performed merely to compare the algorithms' output to the "ground truth" in a simple case. Letting $n = 100$ and $k = 5$ we generate 100 sequences with different levels of noise, denoted ρ, and construct the input matrix C^G based on these. Every algorithm was run 100 times for each value of ρ. A new input matrix was created every round. For RS-GIR, the number of buckets k must be specified in advance. The algorithm was provided with the correct value of k in every case. Results are given in Figure 2. Interestingly RS-GIR behaves very much like the

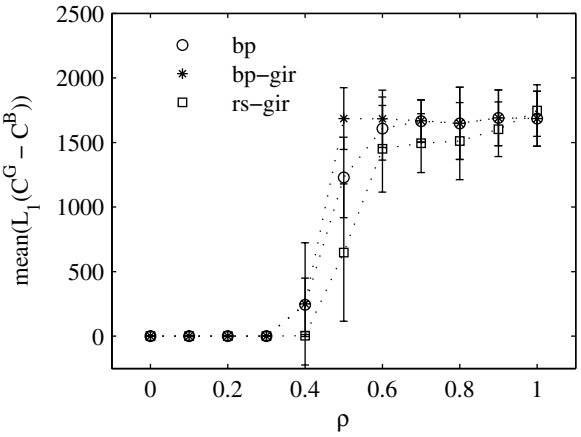

Figure 2: Results for noise tolerance when using artificial data. The graph shows how the level of noise ρ in the input affects the average cost in terms of L_1 between the input matrix C^G ("ground truth") and output matrix C^B for the different algorithms.

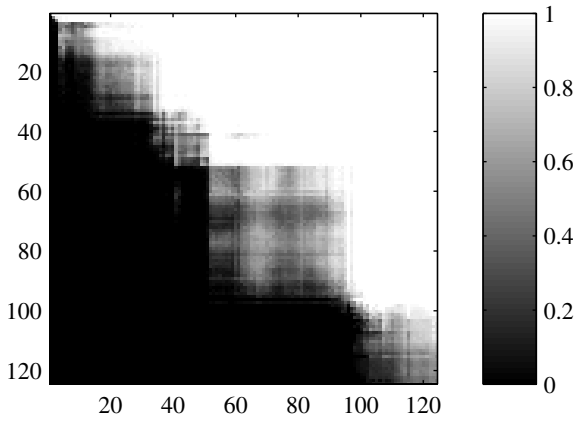

Figure 3: Pair order matrix of the paleontological data. The order for matrix rows/columns is obtained by running the algorithm BP-PR.

pivot-based algorithms. All algorithms are able to sustain up to 30% noise in the input.

6.2 Paleontology Data

The NOW database [14] contains records of fossil discoveries of late Cenozoic land mammals in Europe. The dataset we use contains information on 139 species and 124 fossil discovery sites. For every fossil in the dataset, the site of discovery and the species in question is reported. In general several fossils are found at the same site, thus, the data can be represented as a 0–1 matrix of dimension 124×139. A very important problem in analysing this kind of data in paleontology is that of biochronology: in the lack of geological evidence or geochronologically datable materials, obtain an ordering of the sites based on the sites/species 0–1 matrix.

Bucket orders are a in fact a natural way of representing the temporal order of the site in this case. Determining an exact age for a given site can be very hard. Instead experts have assigned each site to an Mammal Neogene (MN) class, classes ranging from 3 (oldest) to 17 (youngest). Clearly the MN-classification is a bucket order on the sites. One evaluation criterion for the algorithms is how well the discovered bucket orders agree with the bucket order defined by the MN-system.

To run our algorithms on the paleontology data we need to obtain a pair order matrix for the sites based on the original sites/species 0–1 matrix. The pair order matrix is obtained by sampling all possible orderings of the sites with a Markov Chain Monte Carlo method [25]. The test was performed by running all of the three proposed algorithms 100 times on this matrix. Number of buckets k was set to 17 for RS-GIR, since this was the average number of buckets discovered by the BP algorithm.

The resulting pair order matrix of the sites, when the matrix is rearranged according to an order found by the basic pivot algorithm is shown in Figure 3. The bucket-order structure of the sites is evident.

More detailed results for the same input matrix are given in Table 1. The best performance is achieved when using RS-GIR. The segmentation postprocessing step also improves the basic pivot algorithm. Also, the best solution found by BP is almost as good as RS-GIR.

Even with this relatively small dataset BP-GIR and RS-GIR are both several orders of magnitude slower than BP. Therefore it is in fact much faster to run BP many times and use the best solution found. With very large datasets this may be infeasible, however, since computing the error $L_1(C^B, C)$ takes time of order $O(n^2)$.

We also compare the bucket orders produced by the algorithms against the "ground truth" order B_{MN} provided by the MN-classification system. The errors (c_{MN}) are smaller in this case, showing that the bucket orders discovered by our algorithms are in good agreement with the MN system. The algorithm that compares the best against the MN ordering is again RS-GIR. However, basic BP found at least once a solution that is only 0.7 percent worse in terms of c_{MN} than the one found by using RS-GIR.

Very important for this case study is also the number of buckets discovered by the parameter-free algorithms. The number of different MN classes in this data is 14. On the average the pivot based algorithms find 17 buckets. The correspondance is thus very good, given that the exact number of classes in the MN-system is still a matter of debate.

7. RELATED WORK

As mentioned above, the pivot algorithm presented in this paper is inspired by the algorithm presented by Ailon et al. [2] for the feedback arc set problems, i.e., for learning total orders in tournament graphs.

A considerable amount of previous work has focused on ranking query results, either database queries [1, 6, 19, 22], or web search results [4, 5, 17, 20, 10]. This is quite different from our framework since only items related to query are ranked, and usually only the top items of the ranking are of interest.

More related to our approach are data mining papers that attempt to describe the complete set of data using partial orders [23, 27], or fragments of order [16]. Related is also the work on rank aggregation, led by Fagin et al. [11, 12, 13]

	BP	BP-GIR	RS-GIR
avg. c	315.20	278.16	277.08
min. c	287.59	277.05	277.08
max. c	368.00	280.49	277.08
std. c	18.41	0.75	0.00
avg. c_{MN}	356.03	232.64	228.00
min. c_{MN}	229.50	220.50	228.00
max. c_{MN}	490.00	259.00	228.00
std. c_{MN}	65.86	7.56	0.00
avg. N_B	17.54	17.16	17.00
min. N_B	14.00	14.00	17.00
max. N_B	20.00	20.00	17.00
std. N_B	1.43	1.42	0.00

Table 1: Test results for paleontological data ($\beta = 0.25$). Each algorithm was run for 100 times on the input matrix shown in Figure 3. Here $c = L_1(C^B, C)$, where C^B is the solution found by the algorithm and C is the input matrix. Number of buckets in the solution is denoted by N_b and $c_{MN} = L_1(C^B, C^{MN})$, where C^{MN} is a matrix corresponding to the MN-order defined by experts. N_B is the number of buckets in the order returned by the algorithm.

and by Dwork et al. [10], which attempts to combine several total rankings into one single total ranking that agrees with the given rankings as much as possible. The same problem of combining different permutations has also been studied using probabilistic methods [21].

Ranking problems have also been considered in machine learning community under different formulations, for example, in [15] they discuss the problem of learning a total order on labels from a set of examples that declare preference between pairs of labels, and in [8] they consider the problem of learning a ranking function from points in in $\mathbb{R}^d$ to a discrete set of grades $\{1, \ldots, k\}$. Much more related with our work is the paper by Cohen et al. [7]. They suggest the problem of learning an order of items based on a pairwise score function between item pairs, just as our pair order matrix. However, their objective function is different than ours and it does not explicitly model the concept of buckets.

8. CONCLUSIONS

We have described an approach to finding bucket orders from data. Bucket orders are a natural class of partial orders and they are suitable for modeling real-life situations where ties between elements are possible.

We gave a simple algorithm for finding bucket orders and showed that it has a bounded approximation ratio for the NP-complete task of finding the bucket order that best approximates the pair order matrix. The algorithm is scalable and works very fast in practice. We also discussed variations of the method that yield even better approximations of the pair order matrix.

We demonstrated the usefulness of the algorithm on both artificial data and real data on fossils. The results show that the pivot algorithm and its variations provide good and intuitive results.

One interesting direction for extending the basic model in the future is to consider the problem of finding several bucket orders using mixture modeling.

9. REFERENCES

[1] S. Agrawal et al. Automated ranking of database query results. In *CIDR*, 2003.

[2] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *STOC*, 2005.

[3] N. Alon. Ranking tournaments. *SIAM Journal of Discrete Mathematics*, 2006. to appear.

[4] A. Borodin et al. Link analysis ranking: Algorithms, theory, and experiments. *ACM Transactions on Internet Technology*, 5(1), 2005.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[6] S. Chaudhuri et al. Probabilistic ranking of database query results. In *VLDB*, 2004.

[7] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[8] K. Crammer and Y. Singer. Pranking with ranking. In *NIPS*, 2001.

[9] A. B. Cruse. On removing a vertex from the assignment polytope. *Linear Algebra and its Applications*, 26:45–57, 1979.

[10] C. Dwork et al. Rank aggregation methods for the web. In *WWW*, 2001.

[11] R. Fagin et al. Comparing and aggregating rankings with ties. In *PODS*, 2004.

[12] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *SODA*, 2003.

[13] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, 2003.

[14] M. Fortelius et al. Provinciality, diversity, turnover and paleoecology in land mammal faunas of the later Miocene of western Eurasia. In R. Bernor, V. Fahlbusch, and W. Mittmann, editors, *The Evolution of Western Eurasian Neogene Mammal Faunas*, pages 414–448. Columbia University Press, New York, 1996.

[15] J. Fürnkranz and E. Hüllermeier. Pairwise preference learning and ranking. In *ECML*, 2003.

[16] A. Gionis, T. Kujala, and H. Mannila. Fragments of order. In *KDD*, 2003.

[17] T. Haveliwala. Topic-sensitive pagerank. In *WWW*, 2002.

[18] J. Himberg et al. Time series segmentation for context recognition in mobile devices. In *ICDM*, 2001.

[19] I. F. Ilyas et al. Rank-aware query optimization. In *SIGMOD*, 2004.

[20] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), 1999.

[21] G. Lebanon and J. D. Lafferty. Cranking: Combining rankings using conditional probability models on permutations. In *ICML*, 2002.

[22] C. Li et al. Query algebra and optimization for relational top-k queries. In *SIGMOD*, 2005.

[23] H. Mannila and C. Meek. Global partial orders from sequential data. In *KDD*, 2000.

[24] P. McJones. Eachmovie collaborative filtering data set, 1997. `http://research.compaq.com/SRC/eachmovie/`.

[25] K. Puolamäki, M. Fortelius, and H. Mannila. Seriation in paleontological data using Markov Chain Monte Carlo methods. *PLoS Computational Biology*, 2(2):e6, 2006.

[26] N. J. A. Slone. The on-line encyclopedia of integer sequences, 2005. http://www.research.att.com/~njas/sequences/.

[27] A. Ukkonen, M. Fortelius, and H. Mannila. Finding partial orders from unordered 0-1 data. In *KDD*, 2005.

[28] H. S. Wilf. *generatingfunctionology*. Academic Press, 1994. `http://www.math.upenn.edu/~wilf/DownldGF.html`.