Rank Correlation Coefficient Correction by Removing Worst Cases

Martin Krone[1] and Frank Klawonn[1 2]

[1] Department of Computer Science
Ostfalia University of Applied Sciences
Salzdahlumer Str. 46/48, D-38302 Wolfenbuettel, Germany
{ma.krone,f.klawonn}@ostfalia.de
[2] Bioinformatics and Statistics
Helmholtz Centre for Infection Research
Inhoffenstr. 7, D-38124 Braunschweig, Germany

Abstract. Rank correlation can be used to compare two linearly ordered rankings. If the rankings include noise values, the rank correlation coefficient will yield lower values than it actually should. In this paper, we propose an algorithm to remove pairs of values from rankings in order to increase Kendall's tau rank correlation coefficient. The problem itself is motivated from real data in bioinformatics context.

Key words: Rank correlation coefficient, greedy algorithm, graph algorithms

1 Introduction

The motivation for the formal problem we will discuss in this paper comes from biological experiments with the bacterium *pseudomonas aeruginosa*. In these experiments, each of more than 4,000 genes was knocked out and the mutants resulting from the knocked out genes were examined under more than 100 conditions. Conditions are, for instance, different antibiotics in varying concentrations. For each condition we obtain a value, describing the deviation from the "normal condition". Genes that are functionally related are expected to show similar behaviour under the same conditions, especially under those where they should be highly activated (expressed) if they were not knocked out. If we order the deviations from the "normal condition" for each mutant, we can compare these rankings of the conditions. A high correlation between two rankings would be a hint to functionally related genes. However, although many of the genes will play a certain role under almost all conditions, there are some conditions for each gene where it might have no influence. Unfortunately, we do not know which conditions these are for each gene. These conditions can therefore lead to a reduction of the correlation between the genes or mutants.

In order to reduce this effect, we do not consider the correlation with respect to all conditions. For each pair of genes, we are allowed to remove a fixed small number k of conditions, to compute the correlation coefficient. Here we

use Kendall's tau rank correlation coefficient [2]. The task is to remove those conditions that lead to the highest increase of the rank correlation coefficient.

The paper is organized as follows. In Sect. 2, we briefly recall Kendall's tau rank correlation coefficient and show how it can be associated with undirected graphs. Section 3 reformulates our problem as a graph problem and discusses the infeasible brute force solution and a greedy approach. An improved greedy algorithm based on a look-ahead strategy is proposed in Sect. 4. Experimental results are provided in Sect. 5 before the final conclusions.

2 Formalization of the problem

Let x and y be two rankings of length n and both be free of duplicate values (so-called *ties*). Then Kendall's tau rank correlation coefficient can be used to measure the degree of correspondence between x and y. It is defined as

$$\tau = \frac{p_\mathrm{c} - p_\mathrm{d}}{\binom{n}{2}}, \tag{1}$$

where p_c denotes the number of concordant (meaning: in the same order) and p_d the number of discordant (meaning: in the opposite order) among all $\binom{n}{2}$ different pairs. Two pairs (x_i, y_i) and (x_j, y_j) are referred to as concordant if $\operatorname{sgn}(x_i - x_j) = \operatorname{sgn}(y_i - y_j)$ and denoted discordant otherwise. As it is assumed throughout this paper that both rankings are free of ties, every two pairs are either concordant or discordant.

Furthermore, the correspondence between x and y can be represented by an undirected graph by applying the following set of instructions:

1. Create a graph $G = (V, E)$ where V is the set of n nodes labelled $v_1, \ldots, v_n$.
2. For every pair (i, j) s.t. $1 \leq i < j \leq n$, add an undirected edge between v_i and v_j to E if (x_i, y_i) and (x_j, y_j) are discordant.

Theorem 1. *For $-1 \leq \tau \leq 1$, the number of edges in the resulting graph is given by*

$$|E| = \binom{n}{2} \frac{1 - \tau}{2}. \tag{2}$$

Proof. As the number of edges $|E|$ equals the number of discordant pairs and

$$p_\mathrm{c} + p_\mathrm{d} = \binom{n}{2} \tag{3}$$

holds, it follows from (1) that

$$\tau = \frac{\binom{n}{2} - 2|E|}{\binom{n}{2}}. \tag{4}$$

Solving (4) for $|E|$ proves this theorem. □

Thus, the resulting graph is free of edges provided that $\tau = 1$ whereas it will equal the so-called *complete graph* K_n if $\tau = -1$.

3 Reformulation as a graph problem

Recall that we seek to delete a fixed constant $k < n$ of conditions for the pair of rankings we want to compare. This set of conditions has to be selected in such a way that as many discordant pairs as possible are removed in order to increase the rank correlation coefficient.

As the correspondence between two rankings can be represented by an undirected graph, an equivalent task is to delete a k-subset of the nodes from the graph such that as many edges as possible are thereby removed. At first glance, finding the best k-subset might seem to be an easily solvable problem. However, as deleting any node from the graph decreases the degree of all its adjacent nodes, this is actually a considerably more difficult task. In fact, this is a variation of the so-called *node-deletion problem* [4]. However, this problem is usually seen with regard to finding a minimum number of nodes whose deletion results in a subgraph that satisfies a given graph-property. As approaches [1] for those kinds of tasks cannot be applied to our specific problem, we will now examine two generic approaches along with their advantages and disadvantages.

3.1 Bruteforce approach

The most obvious approach to our problem is to determine all $\binom{n}{k}$ subsets of k nodes and select the set whose deletion results in the removal of more edges than any other set. Note that the total number of removed edges does not depend on the order the nodes of a set are deleted in. Thus, it is sufficient to test only one of all $k!$ permutations for each set of k nodes.

While this approach guarantees to find the best set, it will rarely be used in practice unless testing all $\binom{n}{k}$ subsets can be done in reasonable time which will only be possible if both n and $\min(k, n-k)$ are very small. In other cases, one is usually interested in a different approach that requires less steps, but accepts at the same time that less edges in comparison to the bruteforce algorithm might be removed from the graph.

3.2 Using a greedy strategy

Greedy algorithms are based on the idea of choosing the local optimum in each step hoping this will lead to the best overall performance. While there are some problems that can efficiently be solved by this approach, such as creating a minimum cost spanning tree [3], greedy algorithms often only find approximate solutions. Note that the definition of the local optimum depends on the specific field of application. As we seek to remove as many edges as possible from the graph by deleting a given number of nodes, a greedy strategy for this problem is to delete the node with the highest degree in each step.

It is important to realize that the strategy described above may fail to remove the maximum number of edges if more than two nodes are deleted from the graph. We show this is indeed true for $k = 3$ by comparing the bruteforce algorithm to this greedy strategy when applied to the graph in Fig. 1 that is based

on the rankings in Table 1. Initially, the greedy algorithm deletes node v_7 as it has a higher degree than any other node. Consequently, five edges are removed from the graph. Note that all remaining nodes have degree three and the graph is now symmetric with respect to $\{v_1, v_2, v_3\}$ and $\{v_4, v_5, v_6\}$. As each pair of nodes from either set is not adjacent, the greedy algorithm will delete any pair. Thus, a total of eleven nodes are removed from this graph. The bruteforce approach, however, yields a better result. Initially, each pair of nodes of the set $\{v_1, v_2, v_3\}$ is not adjacent and each of these nodes has degree four. Consequently, deleting all nodes of the set $\{v_1, v_2, v_3\}$ in arbitrary order allows the bruteforce algorithm to remove twelve edges from the graph. However, this greedy approach might

Table 1. Pair of rankings that can be represented by the graph in Fig. 1

1	2	3	4	5	6	7
5	6	7	1	3	4	2

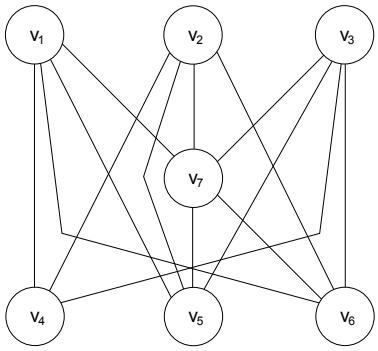

Fig. 1. Example for a graph for which the greedy algorithm removes less edges than the bruteforce approach

still be preferred to the bruteforce algorithm as it can easily be implemented and often comes close to the maximum number of edges that can be removed by deleting a certain number of nodes. As this greedy strategy has to find the node having the highest degree in each of k iterations (which can be done in $O(n)$ steps), its overall complexity is $O(kn)$.

4 An improved greedy strategy

Throughout this section, we use S_i to denote the set of nodes that were removed in the first $i-1$ iterations and let $\deg_S(v)$ be the degree of $v \in V$ if all nodes of the set S were removed from the graph.

The greedy strategy that we described in the previous section does not succeed in removing as many edges as possible from a graph in all cases as it does not take into account the number of edges that can be removed in the subsequent $k-i$ iterations when deciding on the node that will be deleted in the ith step. Thus, in order to improve this greedy strategy for the node-deletion problem, it is necessary to find a way to compute or estimate the number of edges that can be removed in the remaining $k-i$ steps provided that a node v of the set $V \setminus S_i$ was deleted in the ith step. We will now look at three different approaches.

1. One idea is to choose from all $\binom{n-i}{k-i}$ possibilities of selecting $k-i$ nodes from the remaining $n-i$ nodes the set that results in the highest number of removed edges if all its elements are deleted from the current graph. However, this approach basically corresponds to the idea of the bruteforce algorithm described in Sect. 3.1 and will be of limited use for bigger graphs.
2. A different approach is to sort the remaining nodes by their degree in descending order and select the first $k-i$ nodes from the ordered list. Then, approximate the sought-for value by calculating the sum of degrees of the selected nodes. Note that this approach usually overestimates the real number of edges that can be removed in the remaining iterations as it does not consider that some of the selected nodes may be connected by an edge that is consequently counted twice.
3. Moreover, one can also attempt to take into account the effects of deleting v by looking ahead one step in the iteration and apply a second greedy algorithm to determine a lower bound on the number of edges that can be removed in the subsequent iterations. This approach will now be described in more detail.

Assume that the algorithm has completed the first $i-1$ iterations and now has to find the node that will be deleted in the ith step. For each node v of the set $V \setminus S_i$ of the remaining $n-i+1$ nodes, carry out the following steps:

1. Create a copy of the current graph and delete v from this copy.
2. For each of the subsequent $k-i$ iterations, find and delete from the copy the node with the highest degree. Let $h_i(v)$ denote the total number of edges that are removed from the copy in this step.

Finally, set

$$g_i(v) = \deg_{S_i}(v) + h_i(v) \tag{5}$$

to calculate a lower bound on the number of edges that can be removed in the ith step and its subsequent iterations provided that v was deleted in the ith iteration. In order to apply this idea to the node-deletion problem, determine in each iteration $1 \leq i \leq k$ the node $w \in V \setminus S_i$ that maximizes g_i, remove it from the graph along with its adjacent edges and update the set of deleted nodes by setting $S_{i+1} = S_i \cup \{w\}$. A high-level description of this *nested greedy* approach is given in Algorithm 1.

It should be emphasized that $h_i(v)$ provides only a lower bound (but usually a good one provided that $k-i$ is not too large) on the exact number of edges

that can be removed in the remaining $k-i$ iterations if v was deleted in the ith step. While an algorithm deleting nodes on the basis of g_i will consequently not remove as many edges as possible in all cases, it will never perform worse than the greedy algorithm described in Sect. 3.2.

Theorem 2. *The nested greedy algorithm never removes less edges than the greedy approach.*

Proof. If $t \in V$ denotes the node having the highest degree in the initial graph, the total number of edges removed by the greedy algorithm equals $g_1(t)$. Algorithm 1 evaluates $g_1(v)$ for all $v \in V$ to calculate a lower bound on the overall number of edges that can be removed under the assumption that v was deleted in the first step and selects the node that maximizes g_1. As $t \in V$, it follows that $g_1(t) \leq \max_{v \in V} g_1(v)$ and the nested greedy algorithm will therefore never remove less edges in comparison to the greedy approach. □

```
1  S ← ∅
2  r ← 0 // Denotes the number of edges removed by this algorithm
3  for i ← 1 to k do
4  |  b ← null
5  |  m ← −1 // Any other value less than zero is fine
6  |  for v ∈ V \ S do
7  |  |  u ← deg_S(v)
8  |  |  S' ← S ∪ {v} // Create a copy and delete v from this copy
9  |  |  for j ← i + 1 to k do
10 |  |  |  w ← v ∈ V \ S' : deg_S'(v) ≥ deg_S'(x) ∀ x ∈ V \ S'
11 |  |  |  u ← u + deg_S'(w)
12 |  |  |  S' ← S' ∪ {w} // Delete w from the copy
13 |  |  end
14 |  |  if u > m then
15 |  |  |  m ← u
16 |  |  |  b ← v
17 |  |  end
18 |  end
19 |  r ← r + deg_S(b)
20 |  S ← S ∪ {b} // Update graph by deleting b
21 end
```

Algorithm 1: The nested greedy algorithm for the node-deletion problem

The inner loop (lines 9 to 13) of Algorithm 1 that is used to calculate h_i makes this approach more complex in comparison to the greedy algorithm. Recall that the number of nodes that have not been removed from the graph at the beginning of the ith iteration ($1 \leq i \leq k$) equals $n-i+1$. In each iteration i and for every node $v \in V \setminus S_i$ of the set of the remaining nodes a copy of the graph is created which v is deleted from. Then, $k-i$ iterations (i.e. for $i+1 \leq j \leq k$) of the

inner loop are used to calculate $h_i(v)$. In each of these iterations the node having highest degree among all $n - j + 1$ nodes that remain in the copy of the graph has to be found and deleted from the copy. Thus, the total number of steps can be evaluated as

$$\sum_{i=1}^{k} \left((n - i + 1) \cdot \sum_{j=i+1}^{k} (n - j + 1) \right) = O\left(k^2 n^2 + k^4\right). \tag{6}$$

Recall that a total of k nodes are removed from a graph with n nodes. As k will be much smaller than n in our field of application, the total number of steps is limited by $O\left(k^2 n^2\right)$.

Further improvement of Algorithm 1 requires a better estimation of the number of edges that can be removed in the subsequent iterations than the lower bound provided by h_i. We found that increasing the level of nesting the greedy algorithms easily allows for an improved lower bound and will be useful if either the number of nodes in the graph or the number of nodes that one is allowed to remove increases. To illustrate this idea, the entire Algorithm 1 was nested into another greedy algorithm. This new algorithm has running time $O\left(k^3 n^3\right)$ as Algorithm 1 will be called $O(n)$ times in each of k iterations and will be denoted *twice nested greedy*. It can be shown using the arguments of the previous proof that this new algorithm will never perform worse than Algorithm 1.

5 Experimental results

We compared our proposed nested greedy as well as the twice nested greedy algorithm to the bruteforce approach and the greedy algorithm on the basis of three parameters. The first parameter, n, denotes the number of conditions used for the comparison of two rankings. While n was larger than 100 in the experiments with the bacterium pseudomonas aeruginosa, we used values ranging from 30 to 60 to keep the running times of the algorithms, and in particular of the bruteforce approach, low. The second parameter, denoted τ, represents the value of Kendall's tau rank correlation coefficient for a pair of rankings. Recall that the number of edges in a graph is given by $\binom{n}{2}\frac{1-\tau}{2}$ for $-1 \leq \tau \leq 1$. Our tests included values of 0, 0.25 and 0.5 in order to test the algorithms on graphs of varying sparsity. Finally, as k, the number of conditions that we are allowed to remove to increase the rank correlation coefficient, is unknown, we used two values ranging from 10% to 15% of the value of n.

The tests were performed as follows. For each combination of n and τ, we created a list of 10,000 pairs of rankings with a length of n and a rank correlation coefficient of τ. If this was not possible, for example if $n = 15$ and $\tau = 0$, the pairs of rankings were created in such a way that their rank correlation coefficient is as close to τ as possible. Then, each pair of rankings was used to create an undirected graph using the instructions in Sect. 2. Finally, the algorithms were applied to delete k nodes from each graph in such a way that as many edges are thereby removed. The numbers in the three rightmost columns of

Table 2 represent the number of times (out of 10,000) that each of the algorithms removed less edges from a graph in comparison to the bruteforce approach. In

Table 2. Results on comparing three different greedy algorithms with respect to the bruteforce approach

n	τ	k	Greedy	Nested Greedy	Twice Nested Greedy
30	0	3	205	7	0
30	0	5	548	31	3
30	0.25	3	149	6	0
30	0.25	5	466	21	0
30	0.5	3	183	3	0
30	0.5	5	479	33	0
40	0	4	291	12	0
40	0	6	482	43	0
40	0.25	4	190	4	0
40	0.25	6	470	44	3
40	0.5	4	227	4	1
40	0.5	6	483	39	1

order to compare the algorithms on bigger graphs, a slightly different approach was required. For each combination of n and τ, we again created 10,000 pairs of rankings, but excluded the bruteforce approach from the tests as it could not be applied to the resulting graphs in reasonable length of time. As the twice nested greedy algorithm never performs worse than the greedy or the nested greedy approach, all comparisons were no longer done with respect to the bruteforce approach, but to the twice nested greedy algorithm. Consequently, the results in Table 3 only provide a valuable indication but do not exactly resemble how well the greedy and the nested greedy algorithm perform on bigger graphs in comparison to the bruteforce approach. We conclude from these results that the greedy algorithm should not be used for the correction of the rank correlation coefficient as it happens quite frequently that this algorithm does not remove as many discordant pairs as possible, even if the number of conditions that one is allowed to remove is small. Our proposed nested greedy algorithm, however, provides a more reliable method and removed less discordant pairs in comparison to the bruteforce approach in only very few cases. While the twice nested greedy algorithm yields results that come even closer to those of the bruteforce approach, it shall only be used if one accepts the higher running time. However, if both n and $\min(k, n-k)$ are very small, the bruteforce approach might still be used.

6 Conclusions

We have proposed an efficient algorithm to cope with an otherwise infeasible problem. Our basic assumption was that the number k of conditions to be re-

Table 3. Results on comparing the greedy and the nested greedy approach with respect to the twice nested greedy algorithm

n	τ	k	Greedy	Nested Greedy
50	0	5	269	13
50	0	8	612	49
50	0.25	5	207	9
50	0.25	8	548	36
50	0.5	5	226	8
50	0.5	8	601	48
60	0	6	304	15
60	0	9	567	46
60	0.25	6	269	11
60	0.25	9	566	45
60	0.5	6	261	10
60	0.5	9	559	63

moved is fixed. Future work will include investigations on choosing k automatically based on statistical considerations, i.e. to find the point, when we start to increase the rank correlation coefficient artificially.

References

1. Fujito, T.: A Unified Local Ratio Approximation of Node-Deletion Problems (Extended Abstract). In: Algorithms - ESA '96, pp. 167–178. Springer, London (1996)
2. Kendall, M.G.: A new measure of rank correlation. Biometrika 30, 81–93 (1938)
3. Kruskal, J.B.: On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proc. Am. Math. Soc. 7, 48–50 (1956)
4. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: 10th Annual ACM Symposium on Theory of Computing, pp. 253–264. ACM, New York (1978)