Journal of Statistical Software

January 2006, Volume 15, Issue 1. *http://www.jstatsoft.org/*

Efficient Calculation of Jackknife Confidence Intervals for Rank Statistics

Roger Newson
King's College London

Abstract

An algorithm is presented for calculating concordance-discordance totals in a time of order $N \log N$, where N is the number of observations, using a balanced binary search tree. These totals can be used to calculate jackknife estimates and confidence limits in the same time order for a very wide range of rank statistics, including Kendall's tau, Somers' D, Harrell's c, the area under the receiver operating characteristic (ROC) curve, the Gini coefficient, and the parameters underlying the sign and rank-sum tests. A Stata package is introduced for calculating confidence intervals for these rank statistics using this algorithm, which has been implemented in the Mata compilable matrix programming language supplied with Stata.

Keywords: balanced binary search tree, jackknife, confidence interval, rank statistics, Kendall's tau, Somers' D, Harrell's c, ROC area, Gini coefficient, sign test, rank-sum test.

1. Introduction

Rank or so-called "non-parametric" statistics are in fact based on parameters, which are usually equal to zero or 0.5 under a null hypothesis, such as the hypotheses tested by rank-sum tests or by sign tests. Such parameters include Kendall's τ (Kendall and Gibbons (1990)), Somers' D (Somers (1962)), Harrell's c (Harrell, Califf, Pryor, Lee, and Rosati (1982)), the area under the receiver-operating characteristic (ROC) curve (Hanley and McNeil (1982)), and the Gini coefficient, defined as the difference between the areas above and below the Lorenz curve (Cowell (1995)). It is often more informative to calculate confidence limits for these parameters (and their differences) than to calculate P-values alone. Confidence intervals for rank statistics are discussed extensively in Newson (2002), which also introduces a Stata package `somersd` as a unified solution to calculating confidence intervals for all these parameters. At a given time, the current version of `somersd` can usually be downloaded from the Statistical Software Components (SSC) archive, which is located at http://ideas.repec.org/c/boc/bocode/s336401.html. This downloading is usually done using the `ssc` command in Stata. The package includes a manual `somersd.pdf`, which contains the formulas used, and also a range of demonstration examples.

Historically, the main problem with confidence intervals for rank statistics has always been the large amount of computational effort required. As argued in Newson (2002), most rank parameters can be defined in terms of the canonical rank parameter Kendall's τ_a (Kendall and Gibbons (1990)). Confidence interval formulas for Kendall's τ_a appeared as early as 1947 in two consecutive papers in the same issue of *Biometrika*, namely Höffding (1947) (who also published extensively as Hoeffding) and Daniels and Kendall (1947). The second paper illustrated the formulas with a worked example, using a dataset of 30 observations. The calculations required were obviously laborious, even for a dataset as small as this, because the usual formulas for calculating Kendall's τ_a and its standard error involve comparisons of every pair of observations in the dataset, implying a number of operations of order N^2, where N is the number of observations. The availability of computers eased this problem to an extent, and Knight (1966) introduced an algorithm, based on computer sorting, to calculate Kendall's τ_a requiring a time of order $N \log N$. However, this algorithm did not calculate standard errors or confidence limits. A possible standard error formula for Kendall's τ_a is provided by the jackknife method, discussed in Arvesen (1969) and Miller (1974). Using the jackknife together with the Knight algorithm would normally imply using the Knight algorithm one time for each observation (excluding that observation from the dataset), implying a computational time of order $N^2 \log N$. Using the bootstrap (Efron and Tibshirani (1993)) with the Knight algorithm would imply a computational time of order $N_{\text{reps}} N \log N$, where N_{reps} is the number of repeated subsamples. All of these formulas imply a lot of time in large datasets (> 1000 observations), even with today's technology.

An alternative solution was presented by Newson (1987), and used a binary search tree to allow calculation of Kendall's τ_a and its jackknife standard error in a time of order $N \log N$. Initially, the algorithm was implemented in Pascal and intended to input a temporary file output by SAS and to output a temporary file for input by SAS. An improved version of the algorithm was later written in FORTRAN77. These programs were used by the author, but could not be used easily by the typical casual user of a statistical package. However, the appearance of Version 9 of Stata, in May 2005, provided Stata programmers with the C-like compilable matrix language Mata (StataCorp LP (2005)). This enabled the author to incorporate the search tree algorithm into a new version of the `somersd` package, which had previously been written in the interpretable Stata language and used a quadratic-time algorithm.

The search tree algorithm, described below, is entirely generic, and could in principle be implemented as a C plugin by any programmer on the R project with the time and the inclination to do so. However, the Mata solution has the advantage of being integrated seamlessly into a Stata package, and therefore being available immediately to Stata users on all platforms on which Stata is available.

2. Statistical formulas

The family of rank parameters estimated by `somersd` is very comprehensive, including extensions to stratified, clustered and left- or right-censored data. The formulas, and a range of demonstration examples, are documented extensively in the accompanying manual `somersd.pdf`. However, the canonical parameter is Kendall's τ_a, and the computational issues for jackknifing the others are more complicated versions of the computational issues for jackknifing Kendall's τ_a, which we will therefore describe here. However, we will generalize Kendall's τ_a immediately to the case where the variables may be left- and/or right-censored and the observations may have "importance weights". We will assume that observations $(X_i, R_i, Y_i, S_i, W_i)$ are sampled independently from a common population, where, for the ith observation, X_i is the X-value, R_i is the censorship indicator for the X-value, Y_i is the Y-value, S_i is the censorship indicator for the Y-value and W_i is the importance weight. The censorship indicators are numeric, and indicate left-censorship (implying a "true" value at or below the stated value) if negative, right-censorship (implying a "true" value at or above the stated value) if positive, and non-censorship (implying a "true" value equal to the stated value) if zero. We define the censored signed difference

$$\mathrm{csign}(U,P,V,Q) \quad = \quad \begin{cases} 1, & U > V \text{ and } P \geq 0 \geq Q \quad , \\ -1, & U < V \text{ and } P \leq 0 \leq Q \quad , \\ 0. & \text{otherwise,} \end{cases} \tag{1}$$

for numeric values U and V and censorship indicators P and Q. Note that $\mathrm{csign}(U,P,V,Q)$ is equal to $\mathrm{sign}(U-V)$ if $P = Q = 0$, implying no censorship. If we define

$$\begin{array}{rcl} T_{ij} & = & \mathrm{csign}(X_i,R_i,X_j,R_j)\,\mathrm{csign}(Y_i,S_i,Y_j,S_j) \quad , \\ T_{i.} & = & \sum_{j=1}^N W_j T_{ij} \quad , \\ T_{..} & = & \sum_{i=1}^N W_i T_{i.} \quad , \end{array} \tag{2}$$

where N is the sample number, then we can define the population Kendall's τ_a as

$$\tau_{X,R,Y,S} \quad = \quad \mathsf{E}\left(W_i W_j T_{ij}\right) \tag{3}$$

(for $i \neq j$), and estimate it from the sample Kendall's τ_a, defined as

$$\hat{\tau}_{X,R,Y,S} \quad = \quad \frac{T_{..}}{N(N-1)} \quad , \tag{4}$$

and estimate the sampling variance of this estimate as the sample variance of the jackknife pseudovalues

$$\psi_i \quad = \quad \begin{cases} 0\,, & N = 1 \quad , \\ W_i T_{i.}\,, & N = 2 \quad , \\ T_{..}/(N-1) \; - \; (T_{..} - W_i T_{i.})/(N-2)\,, & \text{otherwise,} \end{cases} \tag{5}$$

divided by the sample number N. We see that $\hat{\tau}_{X,R,Y,S}$ is simply the mean of the ψ_i, and that the key to calculating the estimate and its standard error is to calculate the concordance-discordance totals $T_{i.}$. However, if we calculate these as suggested by the definition (2), then we must evaluate each of the T_{ij} individually, implying $N(N-1)$ comparisons, and therefore a computational time of order N^2. The algorithm presented below allows the calculation of all the $T_{i.}$ in a time of order $N \log N$, without calculating the individual T_{ij}, by using a balanced binary search tree.

3. Data structures

In the `somersd` package, the search tree algorithm is contained in a Mata function `tidottree()`, whose code is in the file `tidottree.mata`, distributed with the package in accordance with the open-source principle. `tidottree()` has column-vector input parameters `x`, `y`, `weight`, `xcen` and `ycen`, containing, respectively, the X-values, the Y-values, the weights, the X-censorship indicators, and the Y-censorship indicators. These vectors have the same number of rows, and this number is stored internally in a scalar variable `nobs`. `tidottree()` assumes that the input data matrix defined by these column vectors is sorted in non-descending order of `x`, and starts by defining temporary matrices `xpanel`, with one row per observed X-value and two columns containing the minimum and maximum indices with each X-value, and `yval`, with one row per observed Y-value and one column containing these Y-values in ascending order. (These matrices are created using the very useful Mata functions `panelsetup()` and `uniqrows()`, provided as part of "official Stata".) The numbers of distinct X-values and Y-values are stored in scalars `nxval` and `nyval`, respectively.

The next action is the creation of a balanced binary search tree, with one node for each row index of the Y-value matrix `yval`. Binary search trees and their associated terminology are discussed in Chapter 4 of Wirth (1976), and also in Knuth (1997). A binary search tree that might be created by `tidottree()` is illustrated graphically in Figure 1. In this tree, there are 6 possible Y-values, which (in ascending order) are 1, 2, 3, 5, 8 and 13. As Mata is a matrix language, the natural way to define this search tree is to create a matrix `ytree`, with `nyval` rows and 2 columns. The ith row of the first column contains the left daughter index of i (or zero if there is no left daughter index), and the ith row of the second column contains the corresponding right daughter index (or zero if there is no right daughter index). The search tree is produced by a Mata function `blncdtree()` (distributed in the file `blncdtree.mata`), which calls a second Mata function `_blncdtree()` (distributed in the file `_blncdtree.mata`), which calls itself recursively to build the tree. In the terminology of Wirth (1976), this tree is a perfectly balanced search tree, because, for any index i in the tree, the numbers of nodes in the left and right subtrees of i differ by no more than one, and `yval[`j`] < yval[`i`]` for j in the left subtree of i, and `yval[`j`] > yval[`i`]` for j in the right subtree of i.

Figure 1 contains information on 5 column vectors other than `yval`, namely `sweqlc`, `sweqnc`, `sweqrc`, `swlt` and `swgt`. These column vectors have `nyval` rows, and, together with the column vector `yval` and the matrix of daughter indices, they form the search tree matrix. At most times during execution, these additional arrays contain information about the distribution of weights in some subset Ω of the indices of the main data matrix, which consists of the parallel column vectors `x`, `y`, `weight`, `xcen` and `ycen`. For an index i in the search tree, `sweqlc[`i`]` contains the sum of all the `weight[`j`]` where $j \in \Omega$ and `ycen[`j`] < 0` (denoting left-censorship), `sweqnc[`i`]` contains the sum of all the `weight[`j`]` where $j \in \Omega$ and `ycen[`j`] == 0` (denoting non-censorship), and `sweqrc[`i`]` contains the sum of all the `weight[`j`]` where $j \in \Omega$ and `ycen[`j`] > 0` (denoting right-censorship). (The `==` operator here denotes the equality operator, common to C and Mata.) For the same index i, `swlt[`i`]` contains the sum of all the `sweqlc[`k`]` and `sweqnc[`k`]` for indices k in the left subtree of i, and `swgt[`i`]` contains the sum of all the `sweqrc[`k`]` and `sweqnc[`k`]` for indices k in the right subtree of i. When these equalities hold for a subset Ω of indices, we will say that the search tree represents Ω. In the case of the search tree of Figure 1, if all the `weight[`j`]` are equal to one, then the tree will represent a subset Ω of indices of the main data matrix, of which 13 have a Y-value

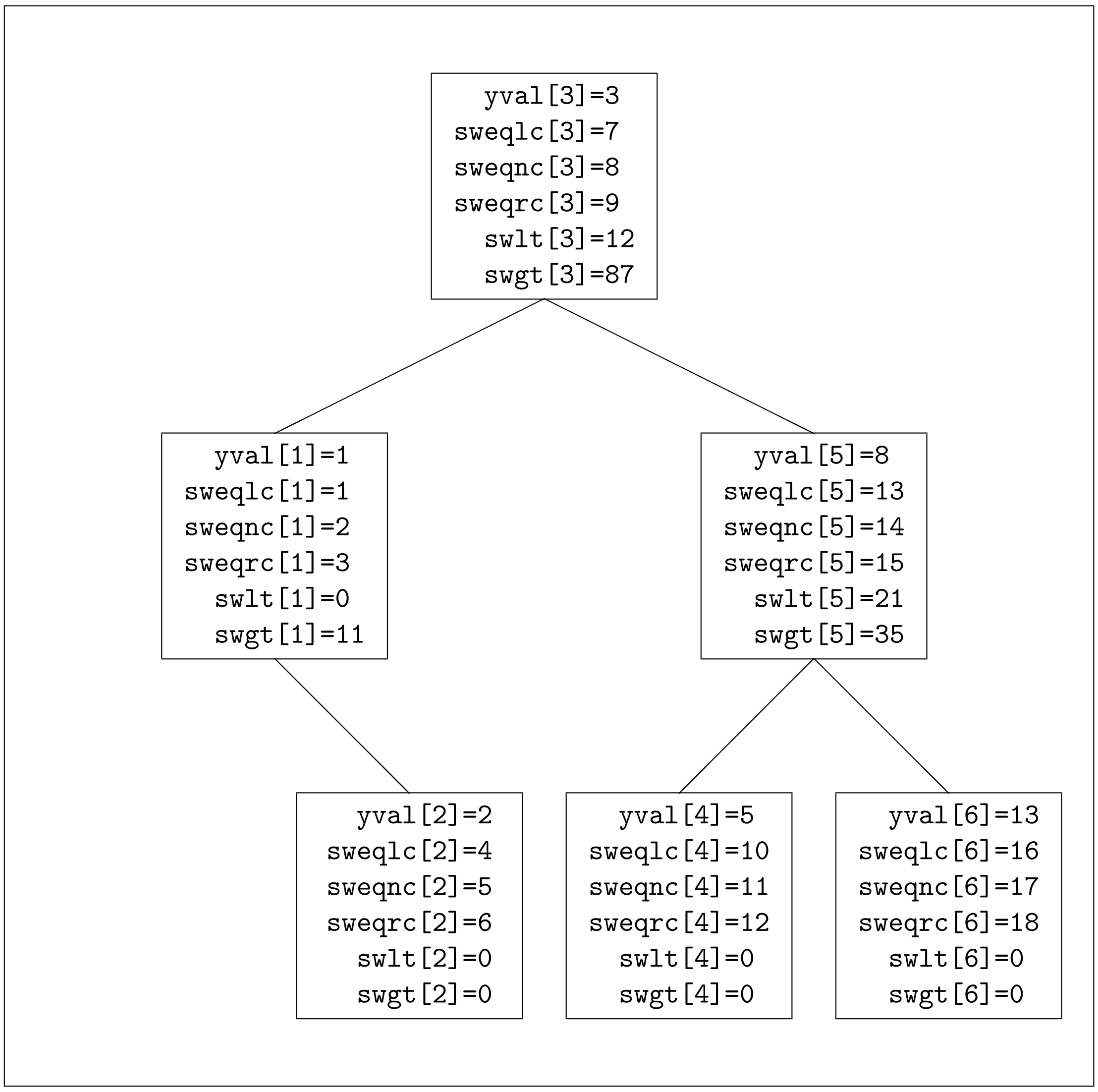

Figure 1: A search tree representing a subset of data matrix indices

left-censored at 8, 14 have a Y-value uncensored at 8, and 15 have a Y-value right-censored at 8. There will also be $21 = 10 + 11$ indices in Ω with Y-values left-censored or uncensored at 5, and $35 = 17 + 18$ indices in Ω with Y-values uncensored or right-censored at 13.

4. The algorithm

We are now in a position to define the two main component operations of the search tree algorithm, which we will denote extraction of left and right subtotals from the search tree matrix for an index of the main data matrix and updating the search tree matrix with an index of the main data matrix. In both cases, we take advantage of the fact that, for each index j of the main data matrix, there exists a path $(h_1, \ldots, h_M)$ of indices of the tree matrix, such that h_1 is the central root index (which can be evaluated by the Mata function `floor()`

as `floor( (nyval + 1 ) / 2)` and which is equal to 3 in Figure 1), `yval[`h_M`]` is equal to `y[`j`]`, and h_{g+1} is the left or right daughter of h_g for $1 \leq g < M$. The operations of subtotal extraction and updating are carried out by iteration along the path.

The left subtotal of an index j of the main data matrix is defined as zero if `ycen[`j`] < 0` (in which case `y[`j`]` is left-censored), and otherwise as

$$\lambda_{\text{tot}}(j) = \sum_{g=1}^{M} \lambda(j, g), \tag{6}$$

where $\lambda(j, g)$ is equal to zero if `y[`j`] < yval[`h_g`]`, to `swlt[`h_g`]` if `y[`j`] == yval[`h_g`]`, and to `swlt[`h_g`] + sweqlc[`h_g`] + sweqnc[`h_g`]` if `y[`j`] > yval[`h_g`]`. Similarly, the right subtotal of j is defined as zero if `ycen[`j`] > 0` (in which case `y[`j`]` is right-censored), and otherwise as

$$\rho_{\text{tot}}(j) = \sum_{g=1}^{M} \rho(j, g), \tag{7}$$

where $\rho(j, g)$ is equal to `sweqnc[`h_g`] + sweqrc[`h_g`] + swgt[`h_g`]` if `y[`j`] < yval[`h_g`]`, to `swgt[`h_g`]` if `y[`j`] == yval[`h_g`]`, and to zero if `y[`j`] > yval[`h_g`]`. In other words, if the search tree represents a subset Ω of indices of the main data matrix, then the left subtotal is the sum of all the `weight[`k`]` for indices k such that

$k \in \Omega$ and csign(`y[`k`],ycen[`k`],y[`j`],ycen[`j`]`) < 0,

and the right subtotal is the sum of all the `weight[`k`]` for indices k such that

$k \in \Omega$ and csign(`y[`k`],ycen[`k`],y[`j`],ycen[`j`]`) > 0.

The operation of updating the tree with an index j of the main data matrix is carried out by iterating along the path $(h_1, \ldots, h_M)$ and incrementing the appropriate stored sums of weights by `weight[`j`]`. For each index h_g on the path, we increment `swlt[`h_g`]` if `y[`j`] < yval[`h_g`]` and `ycen[`j`] <= 0`, we increment `swgt[`h_g`]` if `y[`j`] > yval[`h_g`]` and `ycen[`j`] >= 0`, we increment `sweqlc[`h_g`]` if `y[`j`] == yval[`h_g`]` and `ycen[`j`] < 0`, we increment `sweqrc[`h_g`]` if `y[`j`] == yval[`h_g`]` and `ycen[`j`] > 0`, and we increment `sweqnc[`h_g`]` if `y[`j`] == yval[`h_g`]` and `ycen[`j`] == 0`. If the search tree represented a set Ω of indices of the main data matrix before the update, and $j \notin \Omega$, then the tree will represent the set of indices $\Omega \cup \{j\}$ after the update. Note that the updating operation often involves adding very small values of `weight[`j`]` to large stored numbers. It is therefore very fortunate that Mata does this addition in double precision.

Having defined the component operations of the algorithm, we can now summarize the algorithm as a whole. `tidottree()` first creates the data structures of the search tree and also a column vector `tidot`, with as many rows as the main data matrix, which is initialized to zero and will eventually contain the concordance-discordance totals $T_{i.}$ of Equation (2). The remainder of the algorithm is executed in two stages.

In the first stage, we initialize to zero the tree column vectors `swlt`, `sweqlc`, `sweqrc`, `sweqnc` and `swgt`. The search tree now represents the empty set of indices of the main data matrix. We iterate upwards over all X-values x_{cur} from the lowest to the highest, proceeding for each X-value as follows:

1. For each index j such that `x[`j`] ==` x_{cur} and `xcen[`j`] >= 0`, extract the left subtotal $\lambda_{\mathrm{tot}}(j)$ and the right subtotal $\rho_{\mathrm{tot}}(j)$ from the tree. When this has been done, $\lambda_{\mathrm{tot}}(j)$ will contain the sum of weights for rows of the data matrix known to be below the current row on both the X-axis and the Y-axis, and $\rho_{\mathrm{tot}}(j)$ will contain the sum of weights for rows of the data matrix known to be below the current row on the X-axis and above the current row on the Y-axis. We therefore add $\lambda_{\mathrm{tot}}(j) - \rho_{\mathrm{tot}}(j)$ to `tidot[`j`]`.

2. For each index j such that `x[`j`] ==` x_{cur} and `xcen[`j`] <= 0`, update the tree with index j. When this has been done for all such indices, the tree will represent the set of rows of the data matrix known to be at or below x_{cur} on the X-axis.

In the second stage, we once again initialize `swlt`, `sweqlc`, `sweqrc`, `sweqnc` and `swgt` to zero, causing the tree once again to represent the empty set of indices of the main data matrix, and this time we iterate downwards over all X-values x_{cur} from the highest to the lowest, proceeding for each X-value as follows:

1. For each index j such that `x[`j`] ==` x_{cur} and `xcen[`j`] <= 0`, extract the left subtotal $\lambda_{\mathrm{tot}}(j)$ and the right subtotal $\rho_{\mathrm{tot}}(j)$ from the tree. When this has been done, $\rho_{\mathrm{tot}}(j)$ will contain the sum of weights for rows of the data matrix known to be above the current row on both the X-axis and the Y-axis, and $\lambda_{\mathrm{tot}}(j)$ will contain the sum of weights for rows of the data matrix known to be above the current row on the X-axis and below the current row on the Y-axis. We therefore add $\rho_{\mathrm{tot}}(j) - \lambda_{\mathrm{tot}}(j)$ to `tidot[`j`]`.

2. For each index j such that `x[`j`] ==` x_{cur} and `xcen[`j`] >= 0`, update the tree with index j. When this has been done for all such indices, the tree will represent the set of rows of the data matrix known to be at or above x_{cur} on the X-axis.

When both stages are completed, the column vector `tidot` will contain the concordance-discordance totals $T_{i.}$ of Equation (2), and is therefore returned by `tidottree()` as the result. The above algorithm can be generalized to calculate confidence intervals for Kruskal's γ and Kendall's τ_b (see Goodman and Kruskal (1979)).

5. Performance

The Mata function `tidottree()` has a companion function `tidot()`, distributed with the package in the file `tidot.mata`, which has the same input and output as `tidottree()`, but calculates the $T_{i.}$ using a trivial quadratic-time algorithm, which evaluates the individual T_{ij}. The user has the option of using `tidot()` instead of `tidottree()`, and can therefore compare the performance of the two algorithms. To make the comparison fair, `somersd` does not perform the initial sort by the X-variable (which itself takes a time of order $N \log N$) if the user specifies the quadratic-time algorithm.

The performance trials reported here were carried out on the author's desktop machine, an AT/AT compatible Intel Pentium running at 1.70 GHz with 523,760 Kb RAM, using Intercooled Stata 9.0 under Microsoft Windows 2000. The set task was to generate 2 uniformly-distributed random variables `x` and `y`, and to run the Stata command

```
somersd x y, taua
```

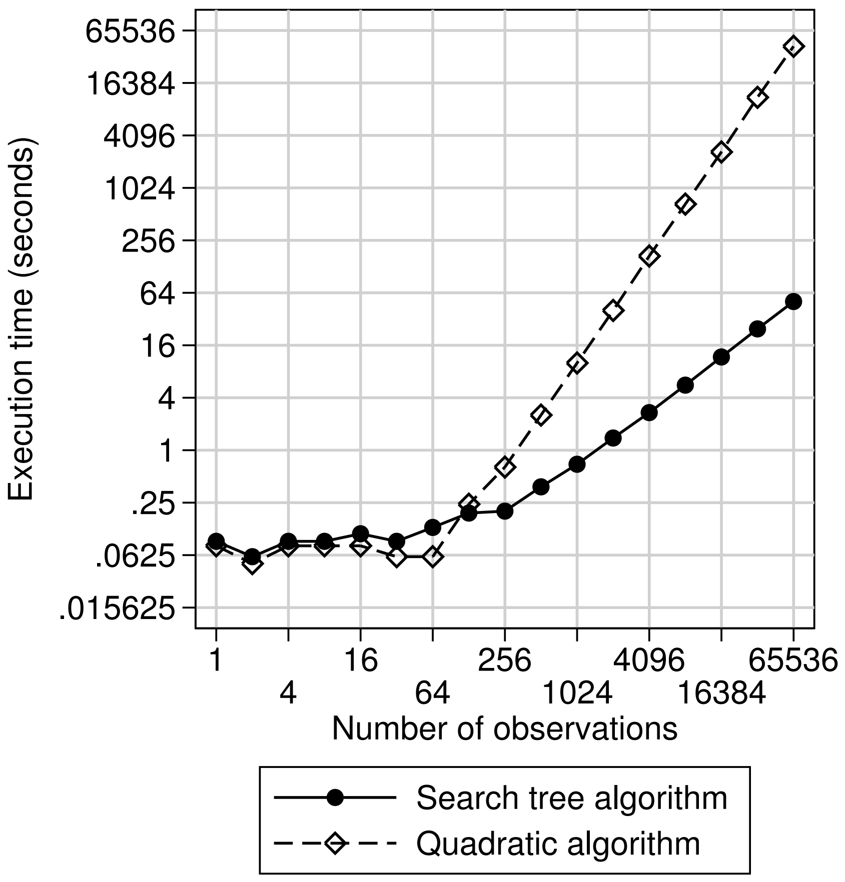

Figure 2: Execution times for search tree and quadratic algorithms

which uses the search tree algorithm to calculate Kendall's τ_a between `x` and `y`, and also Kendall's τ_a between `x` and `x`, which is simply the proportion of pairs of observations with untied values of `x`. For comparison, we also ran the `Stata` command

```
somersd x y, taua notree
```

which does the same calculations using the trivial quadratic-time algorithm. All trials were carried out with the `Stata` setting `memory` set to `16m`, and with the `Stata` setting `rmsg` set to `on`, so that execution times were recorded for each command executed. Numbers of paired observations were set to the values of a binary logarithmic series from $2^0 = 1$ to $2^{16} = 65536$. The results are presented as Figure 2, in which both the number of observations and the execution time are plotted on a binary log scale. With small sample sizes ($N < 100$), execution times seem to be dominated by constant terms which seem to be (if anything) marginally greater for the search tree algorithm. However, for 3-digit or higher sample sizes, the execution time of the quadratic algorithm rises at the predicted gradient of around 2 doublings of execution time per doubling of sample size, whereas the execution time of the search tree algorithm rises at a gradient not much greater than one doubling of execution time per doubling of sample size. The difference in performance becomes spectacular with large datasets ($N > 1000$). At $N = 1024$, the time was .69 seconds for the search tree algorithm and a pregnant pause of 10.02 seconds for the quadratic algorithm. At $N = 8192$, the corresponding times were 5.58 seconds and 667.47 seconds, giving users of the quadratic algorithm a potential excuse for a coffee break. At $N = 65536$, the times were 50.86 seconds (still less than a minute) for the search tree algorithm and 43184.42 seconds (an overnight job of almost 12 hours) for the quadratic algorithm.

Even with today's technology, the new algorithm has been found useful so far by users, known to the present author, who have used it to calculate confidence intervals for the Gini coefficient, Harrell's c and Somers' D on large datasets. Moreover, it is likely that, up until now, many other users (as well as the present author) have been deterred from using these statistical methods on large datasets, or from developing more advanced variants of these methods, because of the computational time taken and/or their unavailability in standard software. Given the tendency for the demand for computing power to rise to fill the available capabilities, and the comprehensive range of clustered and/or stratified and/or censored-data analyses offered by the `somersd` package, the superior performance of the search tree algorithm is likely to continue to be important, at least to some users some of the time. As an example, Harrell, Lee, and Mark (1996) discuss using methods similar to the bootstrap for comparing Harrell's c indices for different prognostic scores on datasets of thousands of subjects. These methods are probably more likely to be used in practice by applied scientists if these scientists have the means to calculate the c index in a time proportional to $N \log N$, rather than to N^2.

References

Arvesen JN (1969). "Jackknifing U-statistics." *Annals of Mathematical Statistics*, **40**, 2076–2100.

Cowell FA (1995). *Measuring Inequality.* Prentice-Hall/Harvester-Wheatsheaf, Hemel Hempstead, UK, 2nd edition.

Daniels HE, Kendall MG (1947). "The Significance of Rank Correlations Where Parental Correlation Exists." *Biometrika*, **34**, 197–208.

Efron B, Tibshirani R (1993). *An Introduction to the Bootstrap.* Chapman & Hall, New York, NY.

Goodman LA, Kruskal WH (1979). *Measures of Association for Cross Classifications.* Springer-Verlag, New York, NY, 1st edition.

Hanley JA, McNeil BJ (1982). "The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve." *Radiology*, **143**, 29–36.

Harrell FE, Califf RM, Pryor DB, Lee KL, Rosati RA (1982). "Evaluating the Yield of Medical Tests." *Journal of the American Medical Association*, **247**(18), 2543–2546.

Harrell FE, Lee KL, Mark DB (1996). "Multivariate Prognostic Models: Issues in Developing Models, Evaluating Assumptions and Adequacy, and Measuring and Reducing Errors." *Statistics in Medicine*, **15**, 361–387.

Höffding W (1947). "On the Distribution of the Rank Correlation Coefficient τ when the Variates are not Independent." *Biometrika*, **34**, 183–196.

Kendall MG, Gibbons JD (1990). *Rank Correlation Methods.* Oxford University Press, Oxford, UK, 5th edition.

Knight WR (1966). "A Computer Method for Calculating Kendall's Tau With Ungrouped Data." *Journal of the American Statistical Association*, **61**(314), 436–439.

Knuth DE (1997). *The Art of Computer Programming. Volume I: Fundamental Algorithms.* Addison-Wesley, Reading, MA, 3rd edition.

Miller RG (1974). "The Jackknife — A Review." *Biometrika*, **61**(1), 1–15.

Newson R (2002). "Parameters Behind "Nonparametric" Statistics: Kendall's Tau, Somers' D and Median Differences." *The Stata Journal*, **2**(1), 45–64. URL http://phs.kcl.ac.uk/rogernewson/papers.htm.

Newson RB (1987). *An Analysis of Cinematographic Cell Division Data Using U-Statistics.* Ph.D. thesis, Sussex University, Brighton, UK.

Somers RH (1962). "A New Asymmetric Measure of Association for Ordinal Variables." *American Sociological Review*, **27**, 799–811.

StataCorp LP (2005). *Stata Statistical Software. Release 9.* Stata Press, College Station, TX. URL http://www.stata.com/.

Wirth N (1976). *Algorithms + Data Structures = Programs.* Prentice-Hall, Inc, Englewood Cliffs, NJ.

Affiliation:

Roger Newson
King's College London
Capital House (5th floor)
42 Weston Street
London SE1 3QD, United Kingdom
E-mail: roger.newson@kcl.ac.uk
URL: http://phs.kcl.ac.uk/rogernewson/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 15, Issue 1 *Submitted:* 2005-08-24
January 2006 *Accepted:* 2005-12-16