Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Performance of Different Objective Functions for Seriation of Pre-Robinson Matrices

Peter A. Cejchan
Institute of Geology, Academy of Sciences of the Czech Republic, v.v.i.

Abstract

Seriation (putting objects to some kind of linear order) is a hard combinatorial problem that must be solved approximately for problems of size approximately > 40. For larger problems an approximate solution, or even optimal solution which, however, is not guaranteed, must be searched using heuristics with some objective function to be optimized. The motivation of this paper is to test performance of different objective functions on (anti-)Robinson matrices using simulated annealing as a metaheuristic. By performance we mean how close to the Robinson form the objective function is able to sort the pre-(anti-)Robinson matrix. Simulated annealing is a probabilistic metaheuristic for global optimization that reveals a good approximation to the global optimum in a large search space. It is often used for approximately solving hard combinatorial problems. We tested performance of 20 objective functions on perfect and imperfect Robinson matrices, in 100 iterations with different initial conditions. The functions are part of the package **seriation**, a Go package implementing seriation of pre-Robinson matrices to Robinson form.

Keywords: seriation, Robinson matrix, similarity matrix, objective function, Go language.

1. Introduction

Seriation means putting objects to some kind of linear order (Hahsler, Hornik, and Buchta 2008; Streng 1991; Niermann 2005; Robinson 1951). For historical overview, see Liiv (2010). For more references on methods, see Cejchan (2014a). In this paper we deal with the objective to seriate a symmetric (dis)similarity matrix as close as possible to the (anti-)Robinson form (see below); block seriation Marcotorchino 1987 and seriation of pre-Petrie (see below) matrices are not covered here. A symmetric (dis)similarity matrix is also known as two-way one-mode data since it has columns and rows (two-way) but only represents one set of objects (one-mode) Hahsler *et al.* 2008. The similarity matrix R is said to be in Robinson form (Robinson matrix, Robinson 1951) iff its elements are non-increasing when moving off the

main diagonal, both horizontally and vertically. Formally, a Robinson matrix, $R = [r_{ij}]$, is a symmetric matrix such that $r_{ij} \leq r_{ik}$ if $j < k < i$ and $r_{ij} \leq r_{ik}$ if $i < j < k$. If rows and columns of a symmetric matrix S can be sorted such that it becomes a Robinson matrix, we call S pre-Robinson. Similarly, the anti-Robinson matrix has non-decreasing elements when moving off the diagonal: $A = [a_{ij}]$ is a symmetric matrix such that $r_{ij} \geq r_{ik}$ if $j < k < i$ and $r_{ij} \geq r_{ik}$ if $i < j < k$. Petrie $P = [p_{ij}]$matrix is an $m \times n$ matrix, whose columns have a single maximum and no other local maxima and the column elements are non-increasing when moving off the column maximum.

Because of its combinatorial nature, as the number of possible solutions grows with problem size (matrix size, n) by the order $O(n!)$, the exact seriation becomes intractable even for quite small matrices, say, $n < 40$ (Hahsler *et al.* 2008) and thus must be solved heuristically for large matrices; e.g., when matrix size $n = 500$, the number of possibilities is $500!/2 = 0.6 \times 10^{1135}$which is enormous. NP-completeness of seriation was proven by Barthelemy and Brucker (2001). For definition of NP-completeness see Garey and Johnson (1979).

The heart of any heuristic is the function to be optimized, the objective function. Although there can be optimization methods specially tailored for every objective function, we used simulated annealing (Kirkpatrick, Vecchi *et al.* 1983; Cerny 1985) as a metaheuristic optimization method for all objective functions, to compare performance under the same conditions.

2. Objective functions

The objective functions are designed either for similarity (Robinson) matrices, or for distance (anti-Robinson) matrices. There are two types of objective functions: gain (also called merit) functions that are to be maximized, and loss functions that are to be minimized.

We also used four functions originally designed for seriation of pre-Petrie matrices. We included these functions because Robinson matrix is a special case of Petrie matrix.

The equations of objective functions to be tested for performance herein follow in the next subsections, where:

D is the distance matrix with elements d_{ij},

S is the similarity matrix with elements s_{ij},

$L(D)$ is the loss function for distance matrix,

$G(D)$is the gain function for distance matrix,

$L(S)$ is the loss function for similarity matrix,

$G(S)$is the gain function for similarity matrix,

w is the window-size defining the range of summation,

$$f(x,y) = \begin{cases} 1 & \text{if } x < y \\ 0 & \text{if } x = y \\ -1 & \text{if } x > y, \end{cases}$$

$$g(x,y) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{otherwise,} \end{cases}$$

and

$$sign(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0. \end{cases}$$

2.1. Anti-Robinson functions

Gain functions

`Wrug(d, p)` returns within row unweighted gradient (WRUG) gain of a permuted distance matrix (Hubert, Arabie, and Meulman 2001, Chapter 4; Brusco 2002: 50, Eq. 6, $g_1(\Psi)$):

$$G(D) = \sum_{k=1}^{n-2} \sum_{l=k+1}^{n-1} \sum_{m=l+1}^{n} sign(d_{km} - d_{kl})$$

`Wrcug(d, p)` returns within row and column unweighted gradient (WRCUG) gain of a permuted distance matrix (Hubert *et al.* 2001, Chapter 4; Brusco 2002: 50, Eq. 7, $g_2(\Psi)$):

$$G(D) = \sum_{k=1}^{n-2} \sum_{l=k+1}^{n-1} \sum_{m=l+1}^{n} sign(d_{km} - d_{kl}) + sign(d_{km} - d_{lm})$$

`Wrwg(d, p)` returns within row weighted gradient (WRWG) gain of a permuted distance matrix (Hubert *et al.* 2001, Chapter 4; Brusco 2002: 50, Eq. 8, $g_3(\Psi)$):

$$G(D) = \sum_{k=1}^{n-2} \sum_{l=k+1}^{n-1} \sum_{m=l+1}^{n} d_{km} - d_{kl}$$

`Wrcwg(d, p)` returns within row and column weighted gradient (WRCWG) gain of a permuted distance matrix (Hubert *et al.* 2001, Chapter 4; Brusco 2002: 50, Eq. 9, $g_4(\Psi)$):

$$G(D) = \sum_{k=1}^{n-2} \sum_{l=k+1}^{n-1} \sum_{m=l+1}^{n} 2d_{km} - d_{kl} - d_{lm}$$

`H(d, p)` returns Szczotka's gain criterion of a permuted distance matrix (Szczotka 1972; Hubert and Schultz 1976; Brusco and Stahl 2000: 201, Eq. 5, Z_5; Brusco, Kohn, and Stahl 2008: 507, Eq. 7, $h(\psi)$):

$$G(D) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} \mid i - j \mid$$

`Ine(d, p)`, returns the inertia gain criterion of a permuted distance matrix (Caraux and Pinloche 2005; Hahsler *et al.* 2008: 5, Eq. 11):

$$G(D) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \mid i - j \mid^2$$

Loss functions

`Lsq(d, p)` returns the least squares loss criterion of a permuted distance matrix (Caraux and Pinloche 2005; Hahsler *et al.* 2008: 5, Eq. 12):

$$L(D) = \sum_{i=1}^{n}\sum_{j=1}^{n}(d_{ij} - \mid i-j \mid)^2$$

`Are(d, p)` returns anti-Robinson events violation loss criterion of a permuted distance matrix (Hahsler *et al.* 2008: 4, Eq. 7, 8; Streng 1991; Streng and Schönfelder 1978; Chen 2002: 2, $AR(i)$; Wu, Tien, and Chen 2010: 773, AR_n)

$$L(D) = \sum_{i<k<j} g(d_{ik}, d_{i,j}) + \sum_{i<k<j} g(d_{k,j}, d_{i,j})$$

`Ware(d, p)` returns weighted anti-Robinson events violation loss criterion of a permuted distance matrix (Hahsler *et al.* 2008; Streng 1991; Tien, Lee, Wu, and Chen 2008; Streng and Schönfelder 1978; Chen 2002:21, $AR(s)$; Wu *et al.* 2010: 773, AR_s):

$$L(D) = \sum_{i<k<j} \mid d_{ik} - d_{i,j} \mid g(d_{ik}, d_{i,j}) + \sum_{i<k<j} \mid d_{kj} - d_{i,j} \mid g(d_{k,j}, d_{i,j})$$

`Dware(d, p)` returns doubly weighted anti-Robinson violation loss criterion of a permuted distance matrix (Hahsler *et al.* 2008; Streng 1991; Tien *et al.* 2008; Chen 2002: 21, $AR(w)$; Wu *et al.* 2010: 773, AR_w):

$$L(D) = \sum_{i<k<j} \mid j-k \mid\mid d_{ik} - d_{i,j} \mid g(d_{ik}, d_{i,j}) + \sum_{i<k<j} \mid i-k \mid\mid d_{kj} - d_{i,j} \mid g(d_{k,j}, d_{i,j})$$

`Gar(d, p, w)` returns generalised anti-Robinson violation loss criterion of a permuted distance matrix (Chen 2002; Tien *et al.* 2008; Wu *et al.* 2010: 773, $GAR(w)$)

$$L(D) = \sum_{(i-w)\leq j<k<i} g(d_{ik}, d_{i,j}) + \sum_{i<j<k\leq(i+w)} g(d_{ij}, d_{ik})$$

reformulated as

$$L(D) = \sum_{(j-w)\leq i<k<j} g(d_{ik}, d_{i,j}) + \sum_{(j-w)\leq i<k<j} g(d_{k,j}, d_{i,j})$$

in Kostopoulos and Goulermas (2014).

`Rgar(d, p, w)` returns relative generalised anti-Robinson violation loss criterion of a permuted distance matrix(Chen 2002; Tien *et al.* 2008)

$$\frac{\sum_{(i-w)\leq j<k<i} g(d_{ik}, d_{i,j}) + \sum_{i<j<k\leq(i+w)} g(d_{ij}, d_{ik})}{\sum_{(i-w)\leq j<k<i} 1 + \sum_{i<j<k\leq(i+w)} 1}$$

reformulated as

$$L(D) = \frac{\sum_{(j-w)\leq i<k<j} g(d_{ik}, d_{i,j}) + \sum_{(j-w)\leq i<k<j} g(d_{k,j}, d_{i,j})}{nw(w-1) - \frac{2w}{3}(w^2-1)}$$

in Kostopoulos and Goulermas (2014).

`Ham(d, p)` returns the loss criterion as the length of the shortest Hamiltonian path (open traveling salesman problem, oTSP) through a permuted distance matrix (Caraux and Pinloche 2005; Hahsler *et al.* 2008: 4 , Eq. 10; Chen 2002: 2, MS):

$$L(D) = \sum_{i=1}^{n-1} d_{i,i+1}$$

`Nsd(d, p)` returns the von Neumann stress loss criterion of the matrix, here applied to a permuted distance matrix, so that $m = n$ (Niermann 2005: 42):

$$L(D) = \sum_{i=1}^{n}\sum_{j=1}^{m}\left[\sum_{k=max(1,i-1)}^{min(n,i+1)}(d_{ij}-d_{kj})^2 \sum_{l=max(1,j-1)}^{min(m,,j+1)}(d_{ij}-d_{il})^2\right]$$

`Msd(d, p)` returns the Moore stress loss criterion of the matrix, here applied to a permuted distance matrix, so that $m = n$ (Niermann 2005: 42, Eq. 1, 2; Hahsler *et al.* 2008: 6, Eq. 15):

2.2. Both Robinson and anti-Robinson functions

Loss functions

`Par(d, p)` returns the sum of squared residues of two parabolas fitted to the rows of a permuted similarity or distance matrix as the loss criterion. This is a new objective function:

$$L(D) = \sum_{i=1}^{n}(d_{ij} - y_{ij})^2$$

where $y_{ij}$is the value of the fitted parabola. First, the position of row maximum is found, and then two parabolas are fitted, one for the left part of the row including maximum, other for the right part, including maximum.

2.3. Robinson functions

Loss functions

`Psis(s, p)` computes energy Ψ_π of the matrix, here applied to a permuted similarity matrix (Podani *et al.* 1994; Miklós, Somodi, and Podani 2005, Eq. 4)

$$L(S) = \sum_{i=1}^{m}\sum_{j=1}^{n} s_{ij}\left(\mid \frac{ni}{m} - j \mid + \mid \frac{mj}{n} - i \mid\right)$$

$$L(D) = \sum_{i=1}^{n}\sum_{j=1}^{m}\left[\sum_{k=max(1,i-1)}^{min(n,i+1)}\sum_{l=max(1,j-1)}^{min(m,,j+1)}(d_{ij}-d_{kl})^2\right]$$

`Bers(s, p)`, returns Bertin's classification criterion of the matrix, here applied to a permuted similarity matrix (Bertin 1981; Pilhofer, Gribov, and Unwin 2012: 2509, eq. 1,$B'(A)$, BCC):

$$L(S) = \sum_{i=2}^{n}\sum_{j=1}^{m-1} s_{ij}\left(\sum_{k=1}^{i-1}\sum_{l=j+1}^{m} s_{kl}\right)$$

3. Optimization method

We used simulated annealing (SA) as an optimization method. simulated annealing is a probabilistic metaheuristic for global optimization that reveals a good approximation to the global optimum in a large search space. It was introduced by Kirkpatrick *et al.* (1983) and independently by Cerny (1985). It is one of many metaheuristics ispired by the Nature; in a condensed matter physics, annealing is a method of obtaining low energy states of solid matter by carefully cooling a melt. SA is a local search metaheuristic belonging to a class called "threshold acceptance algorithms" (Van Laarhoven and Aarts 1987). It contains a stochastic component which enables it to escape from local minima. Therefore, it is often used for approximately solving hard combinatorial problems. Its performance has been reported less dependent on the specific "objective function landscape topology" than some other methods (Van Laarhoven and Aarts 1987). The method is outlined e.g., in Pirlot (1996).

To apply SA to a specific problem, the following parameters must be specified: the state space, the energy (objective) function, the candidate state generator, the acceptance probability function, the annealing schedule, and the initial temperature. Choice of parameters may have significant impact on the method's performance. Unfortunately, there is no general way to find the best choices for a given problem. We tuned the parameters one by one to get an acceptable combination of these, in terms of performance of all tested objective functions on small pre-Robinson matrices.

We used the following values:

- state space was all possible permutations,
- objective functions were as specified above,
- the candidate state generator is described below,
- the acceptance probability function is described below,
- temperature at which to stop = 10^{-8},
- value at which to stop immediately = $-\infty$,
- maximum number of consecutive rejections = 1000,
- maximum number of tries within one temperature = 300,

- maximum number of successes within one temperature = 20,
- Boltzmann constant = 1.0,
- the annealing schedule was $t_{new} = 0.8t$, where t is the current temperature,
- initial temperature = 1.0.

We used the `func proposePerm(p IntVector)` function as the candidate generator. It uses 14 mutation operators inside (Figure 1), using random positions $1 < a < b < c < d < n$:

`p.InvertFrom(a)`: from position a including, reverse the order of elements of the parent permutation vector;

`p.InvertTo(a)`: from the beginning to position a including, reverse the order of elements of the parent permutation vector;

`p.InsideOut(a)`: from the beginning to position a including, reverse the order of elements of the parent permutation vector and from position $a+1$ including, reverse the order of elements of the parent permutation vector;

`p.Scramble3(a)`: from position a including, scramble 3 elements randomly;

`p.Scramble4(a)`: from position a including, scramble 4 elements randomly;

`p.Swap(a, b)`: swap elements at positions a and b; exchange mutation operator (Banzhaf 1990, Figure 2b); swap mutation operator (Oliver, Smith, and Holland 1987); point mutation operator (Ambati, Ambati, and Mokhtar 1991); reciprocal exchange mutation operator (Michalewicz 1992); order based mutation operator (Syswerda 1991).

`p.InvertFromTo(a, b)`: reverse order of the segment between positions a and b including; simple inversion mutation operator (Holland 1975; Grefenstette 1987).

`p.InsertAt(a, b)`: remove element at position a and place it to position b; insertion mutation operator (Fogel 1988; Michalewicz 1992); position based mutation operator (Syswerda 1991)

`p.InvertHeadAndTail(a, b)`: reverse order from position 1 to a including, and from b to n including.

`p.TwoPointSwapInv(a, b)`: like `p.InvertHeadAndTail(a, b)`, but swap the inverted segments; Two Point Swapped Inversion, inspired by Sallabi and El-Haddad (2009): 531.

`p.Displace(a, b, c)`: take segment from a to $b-1$ and insert it at $b+1$; modified from displacement mutation operator (Michalewicz 1992) = cut mutation (Banzhaf 1990).

`p.DisplaceInv(a, b, c)`: like `p.Displace(a, b, c)`, but first invert the first segment.

`p.ThreePointExch(a, b, c)`: Three Point Exchange was inspired by 3-opt move for the traveling salesman problem (TSP; Merz and Freisleben 1997): leave unchanged up to a inclusive; go from a to c and inverse-continue to $b+1$; go from $b+1$ to $a+1$ and continue to b; go from $c+1$ to n.

`p.FourPointExch(a, b, c, d)`: Four Point Exchange was inspired by non-sequential 4-change for the TSP(Merz and Freisleben 1997): leave unchanged up to a inclusive; go from a to $c+1$ and continue to d; go from d to $b+1$ and continue to c; go from c to $a+1$ and continue to b; go from b to $d+1$ and continue to n.

These operators were applied randomly with equal probability.

We used the following function as the acceptance probability function:

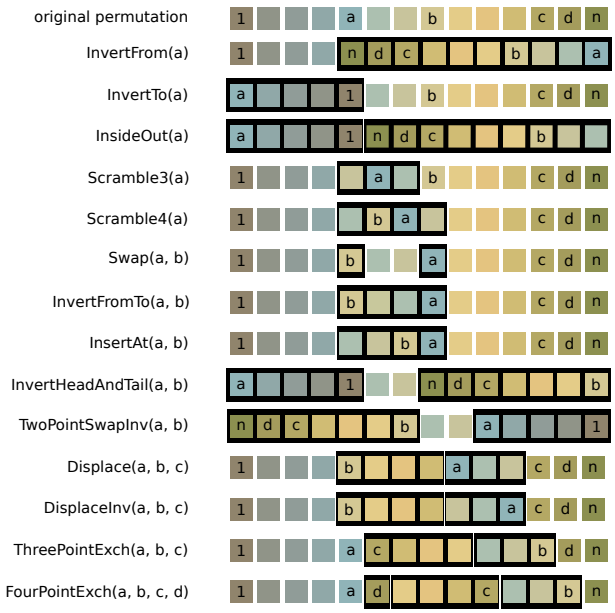

Figure 1: Candidate generator functions; black areas denote the parts of permutation vector affected.

when $E_b - E_n > 1e-6$, where E_b is the best energy so far, and E_n is the candidate's energy, the candidate was accepted; otherwise if $random() < e^{-\frac{E_b - E_n}{kt}}$, the candidate was accepted; otherwise, the candidate was rejected.

4. Implementation

All functions and testing programs were written in Go (http://www.golang.org), a modern programming language that inherits much from C, but implements useful features like automatic memory management, type safety, some dynamic-typing capabilities, additional built-in types such as variable-length arrays and key-value maps. As the original documentation says: *"The* Go *programming language is an open source project to make programmers more productive.* Go *is expressive, concise, clean, and efficient. Its concurrency mechanisms make it easy to write programs that get the most out of multicore and networked machines, while its novel type system enables flexible and modular program construction.* Go *compiles quickly to machine code yet has the convenience of garbage collection and the power of run-time reflection. It's a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language".*

All functions that are covered by the R package **seriation** (Hahsler *et al.* 2008) were tested against that package. Other functions were tested against Kostopoulos and Goulermas (2014).

All the computer code and data can be found at

https://code.google.com/p/seriation/ (Cejchan 2014b),

and the machine-generated documentation for the current release can be found at

http://godoc.org/code.google.com/p/seriation.

5. Performance tests

We tested performance of the 20 objective functions described above on Robinson and "imperfect" Robinson matrices of sizes $n = 50$, 100, 150, 250, and 500. By "imperfect" we mean that a small amount of noise was introduced so that the matrix contained Robinson form violations. Most of the objective functions are designed for anti-Robinson matrices. For those, we converted similarity matrix to dissimilarity one by subtracting each element from the matrix maximum value. Four functions are originally designed for Petrie matrices; in this case, we used the same permutation vector for rows and for columns. For generalized functions that use window, we used window sizes $w = 0.25n$, $0.50n$, and $0.75n$.

Each matrix has been randomly permuted in 100 iterations for each objective function.

Because the resulting permutation vector can be read from both sides, we applied order reversion when needed. "When needed" means when Spearman rank correlation coefficient:

$$\rho = \frac{\sum_i (xi - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (xi - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

with the "known" permutation $p^{known} = [1, 2, ..., n]$ was negative.

The performance was measured in several ways:

- as the Spearman rank correlation coefficient, ρ (see above), with the "known" permutation p^{known}, and its standard deviation,
- as proportion of "perfect hits", i.e., $p^{seriated} = p^{known}$,
- and as proportion of "rank hits", i.e., cases when $p_i^{seriated} = p_i^{known}$,
- as "rank matrix" $R = [r_{ij}]$, where r_{ij} is the count of cases when $p_i^{seriated} = j$,
- and as "pair-order violation matrix" ,$V = [v_{ij}]$ vhere v_{ij} is the count of cases when $p_i^{seriated} > p_j^{seriated}$, if $p_i^{known}, < p_j^{known}$, and $p_i^{seriated} < p_j^{seriated}$, if $p_i^{known}, > p_j^{known}$.

More formally,

$r_{ij} = \sum h(p_i^{seriated}, j)$,

where

$$h(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise,} \end{cases}$$

and

$v_{ij} = \sum g(g(p_i^{seriated}, p_j^{seriated}), g(p_i^{known}, p_j^{known})) + g(g(p_j^{seriated}, p_i^{seriated}), g(p_j^{known}, p_i^{known}))$.

We also measured elapsed time for 100 iterations. Computing times shown in the plots are approximate for Intel® Xeon® CPU E31240 @ 3.30GHz with Linux OS version 2.6.32-431.5.1.el6.centos.plus.x86_64.

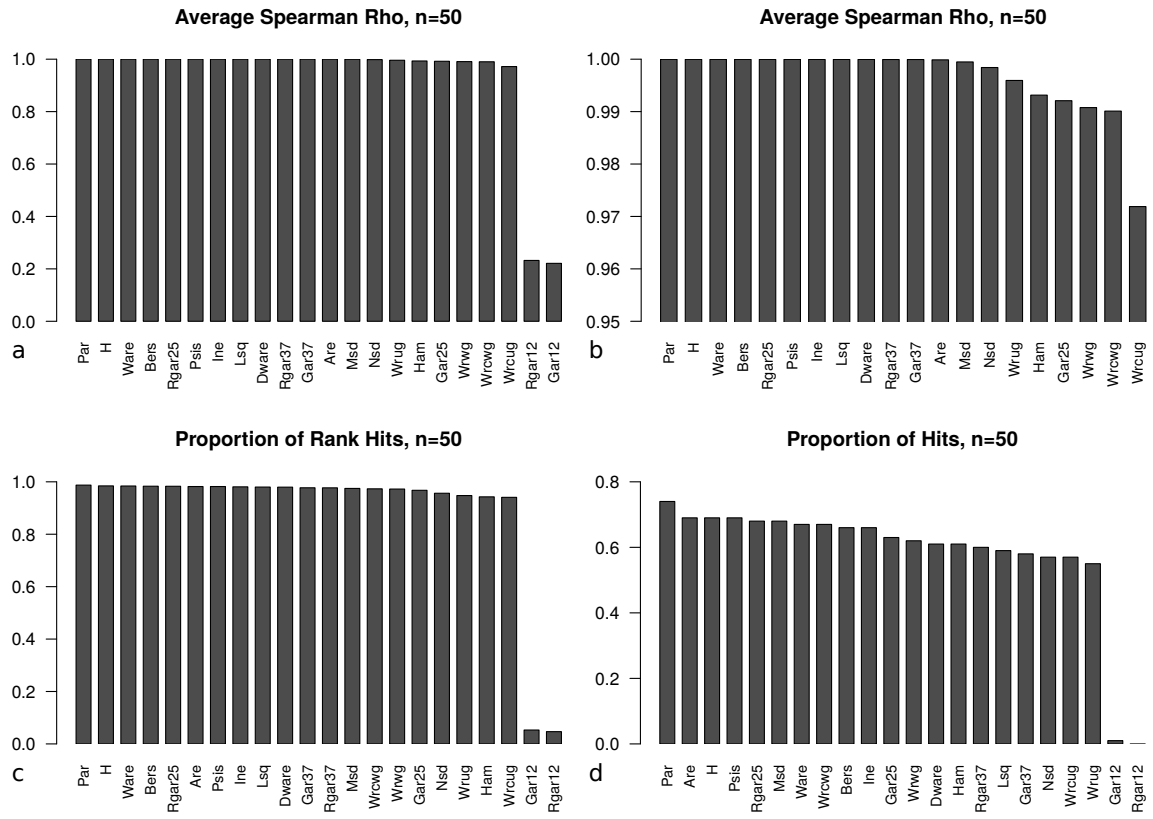

Figure 2: Performance of all tested objective functions on 50×50 Robinson matrix randomly permuted in 100 iterations, sorted by average ρ (a, b), proportion of rank hits (c), and proportion of hits (d). Note that Gar and $Rgar$ performed poorly with window size $w = 0.25n$. Note the range of y-axis in (b).

6. Results

6.1. Robinson matrices

$n = 50$ matrix

For $n = 50$, all functions performed fine, except of those with $w = 0.25n$(see Figure2 . These functions were excluded from further computations. $Bers$, although performing well, took too long time to be computed for higher n (see Figure 3). It was excluded from computations for $n \geq 150$.

$n = 100$ matrices

For $n = 100$, Msd, Nsd, Ham, and $Wrug$ started to perform worse (measured using average Spearmanρ), along with $Rgar$ and Gar with window size $w = 0.25n$, that performed poorly already for $n = 50$. When measured as proportion of "rank hits", additional functions dropped in performance, namely $Rgar$ with window sizes $w = 0.50n$ and $w = 0.75n$. These are the functions that also slightly dropped in performance measured using average Spearmanρ.

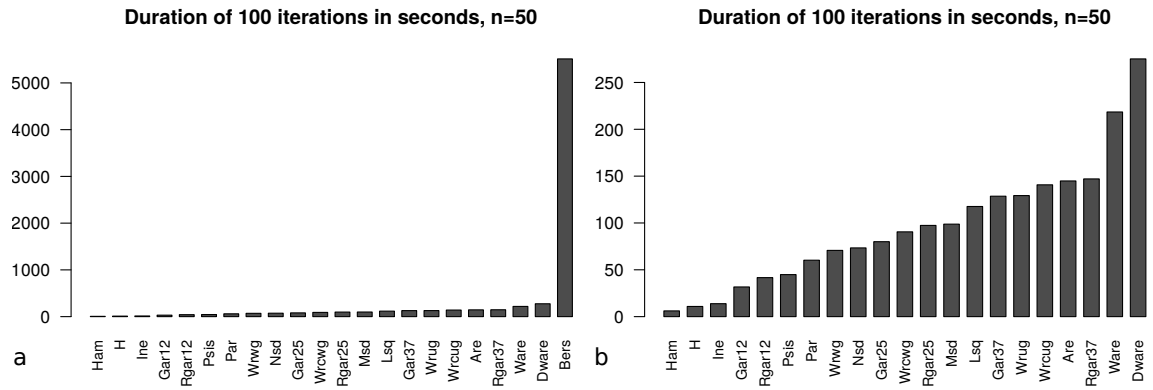

Figure 3: Duration of 100 iterations in seconds, for 50×50 Robinson matrix; for all functions (a), and with *Bers* omitted (b). *Bers*, although performing well, cosumed enormous computing time (approximately 500 times that of H).

Proportion of "perfect hits" dropped below 0.25 for all functions.

n = 150 matrices

For $n = 150$, 12 functions performed with $\rho \geq 0.99$(Figure 5). *Rgar* and *Gar* with $w = 0.25n$ were excluded from further computations. Proportion of "rank hits" separated quite distinctly the group of functions that for larger n started to perform poorly when measured using ρ. Proportion of "perfect hits" dropped to zero for $n \geq 150$.

n = 250 matrices

For $n = 250$, only 10 functions that were computationally feasible performed with $\rho \geq 0.95$(Figure 7). However, when performance was measured as proportion of "rank hits", it dropped rapidly for all functions, below 0.07. Order of functions sorted by time performance did not change.

n = 500 matrices

For $n = 500$, of 16 functions that were computationally feasible 12 performed with $\rho \geq 0.7$ (see Figure 8). Remaining functions were either too much time consuming (thus excluded from computations), or performed poorly, performed with $\rho < 0.4$

6.2. Imperfect Robinson matrix

For $n \leq 150$, there were almost no differences in performance compared to the results for Robinson matrices. Surprisingly, the selected functions that performed well for 250×250 and 500×500 Robinson matrices, also performed almost equally well on 250×250 and 500×500 imperfect Robinson matrices. It means that they are robust to small violations of

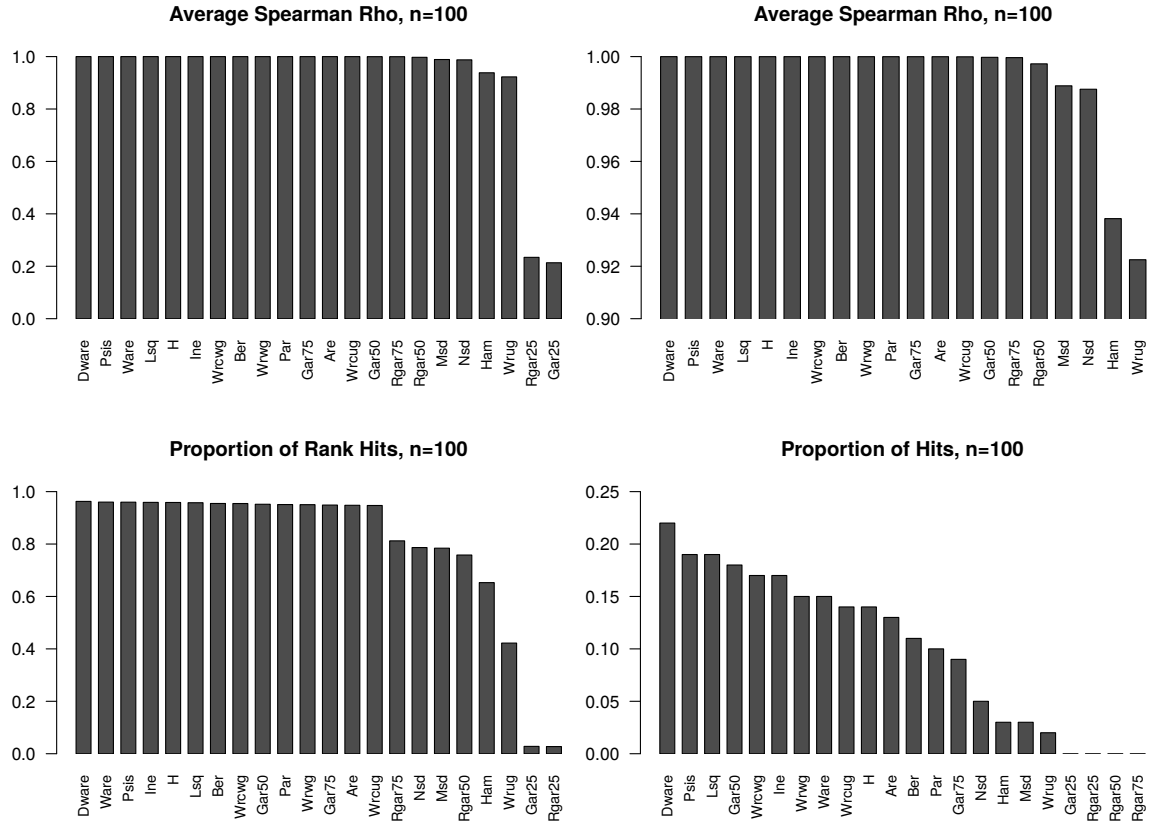

Figure 4: Performance of objective functions on 100×100 matrix, 100 iterations; average Spearman ρ for all functions (a); note that $Rgar$ and Gar with $w = 0.25n$ had poor performance; average Spearman ρ with $Rgar$ and Gar with $w = 0.25n$ omitted (b); proportion of "rank hits" (c); proportion of "hits" (d).

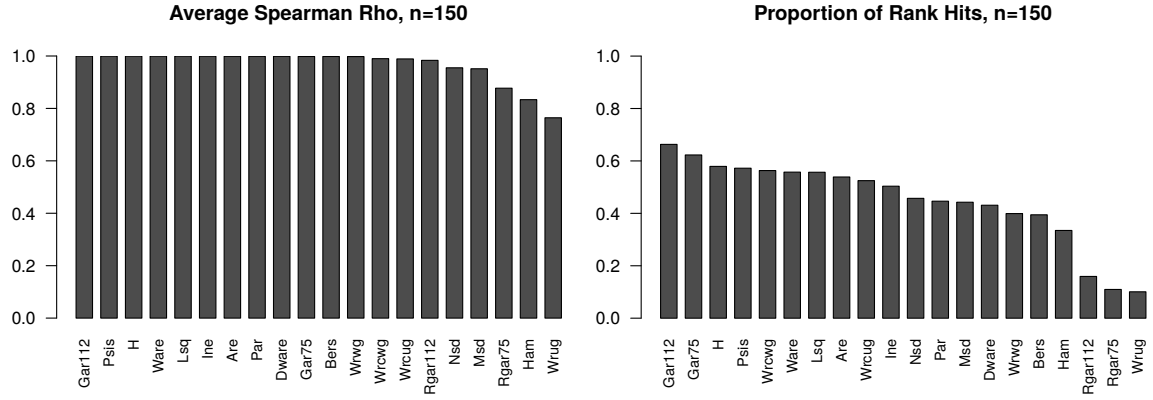

Figure 5: Performance of objective functions on 150×150 Robinson matrix, 100 iterations.

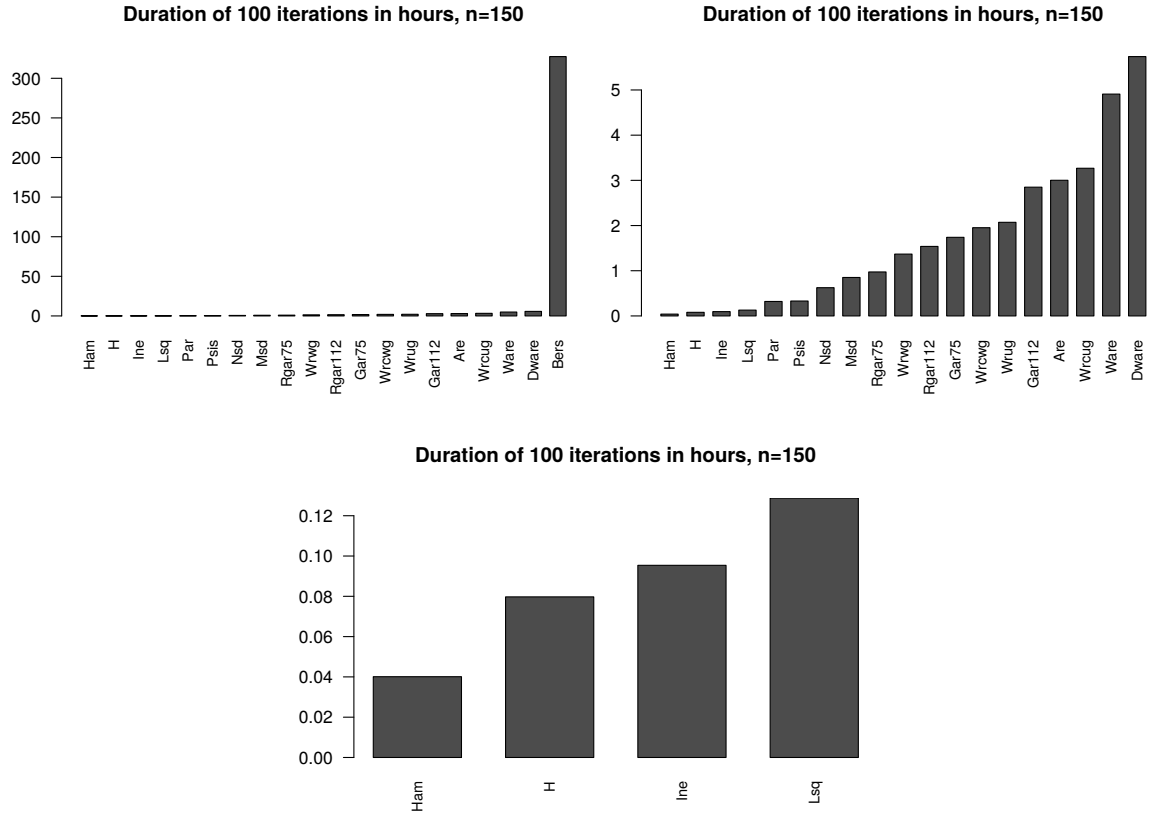

Figure 6: Duration of 100 iterations in hours, for 150×150 Robinson matrix. Note that *Bers* had extremely long computing time.

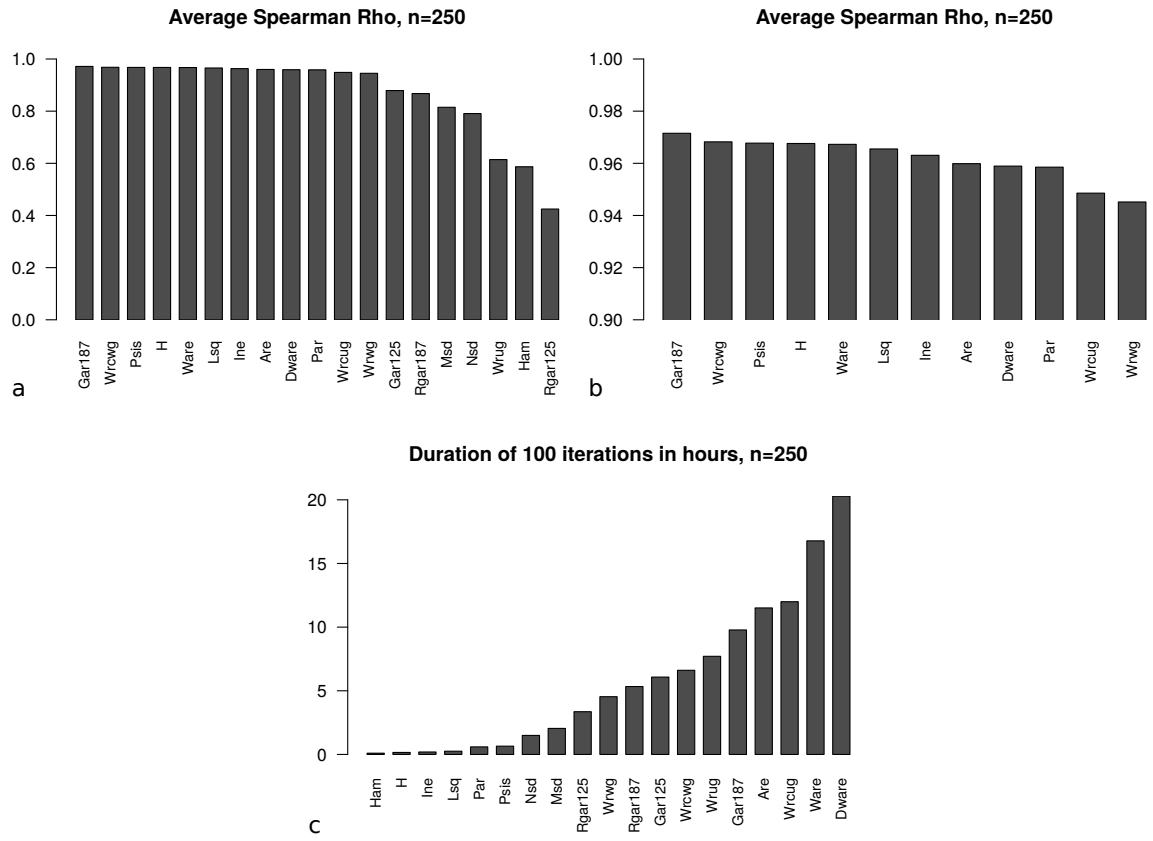

Figure 7: Performance of computationally feasible best objective functions (a, b), and duration of 100 iterations in hours (c) for 250×250 Robinson matrix.

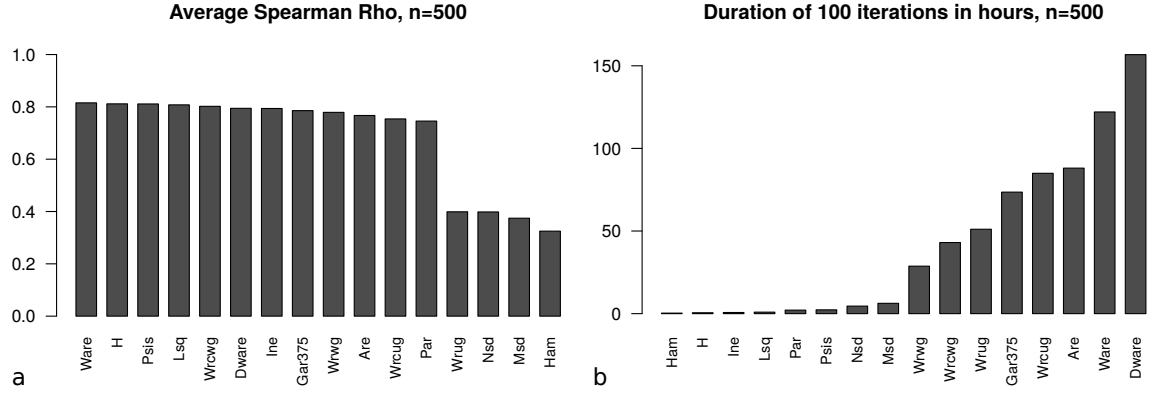

Figure 8: Performance of computationally feasible best objective functions (a), and duration of 100 iterations in hours (b) for 500×500 Robinson matrix.

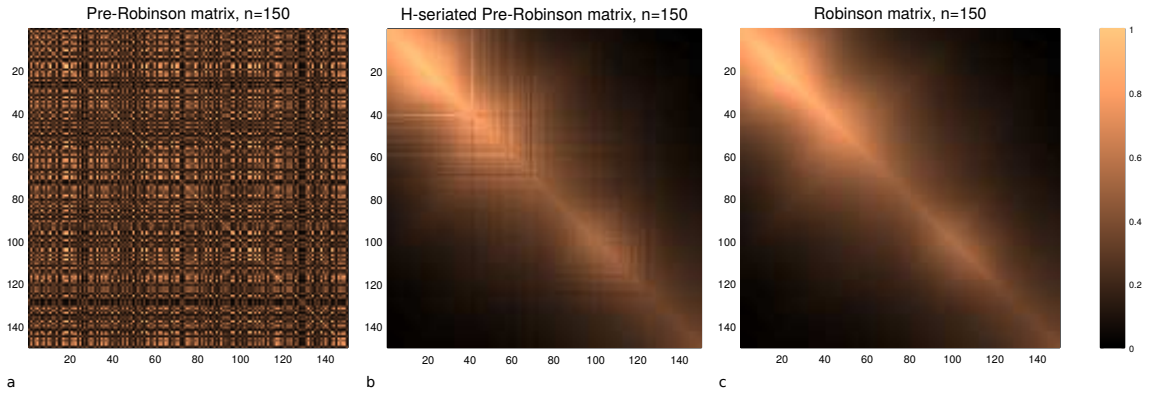

Figure 9: Heatmaps: a pre-Robinson 150 × 150 matrix (a), the same matrix seriated using H in a single iteration (b) the same matrix in Robinson form (c). The seriation produced a suboptimal, but good, solution.

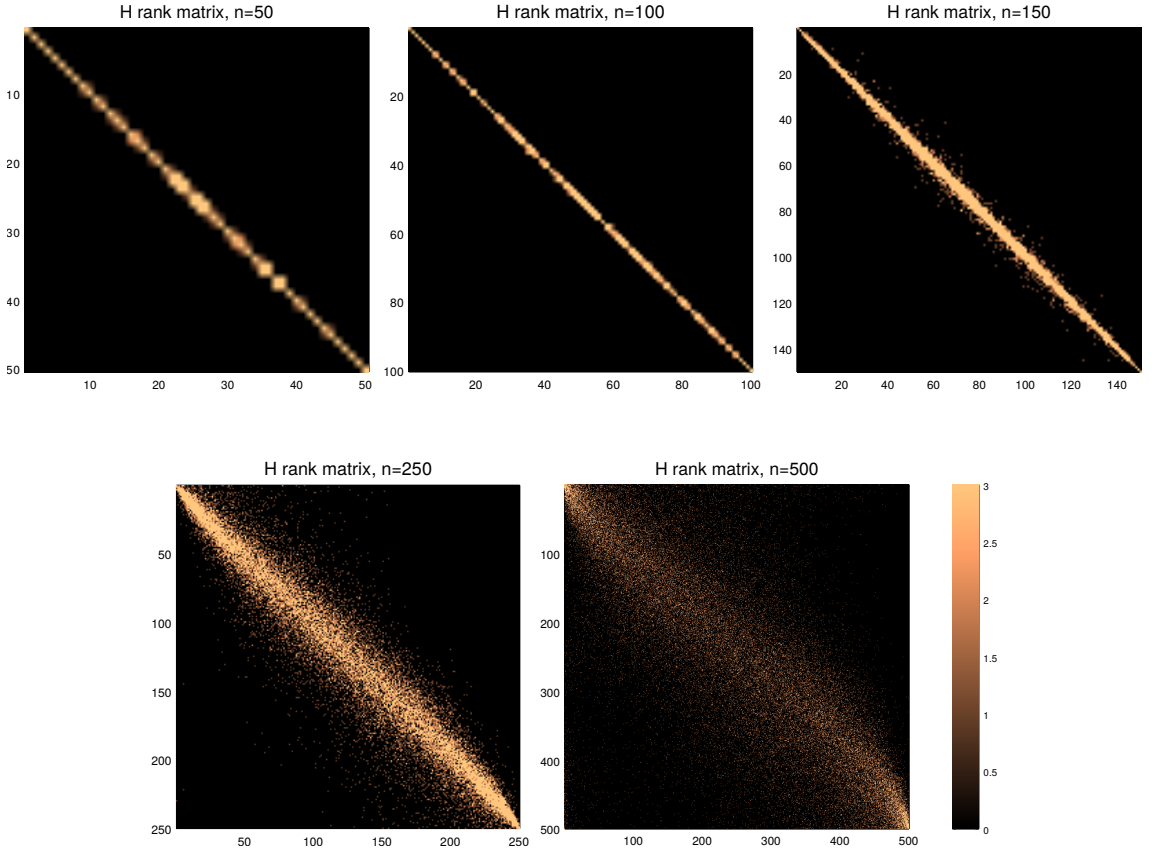

Figure 10: Degradation of performance of the H objective function with growing n. Heatmaps show rank matrices; ideally, all non-zero values should lie on the main diagonal.

Figure 11: Degradation of performance of the H objective function with growing n. Heatmaps show pair-order violations matrices; ideally, there should be no non-zero values. Suboptimally, non-zero values should lie close to the main diagonal.

Robinson form. For 500×500 matrix, $Ware$ performed the best, as was the case also with the Robinson matrix, however, required 225 times more computing time than the second best objective function, H see Figure 13.

7. Conclusions

- Although all objective functions were designed to reach their extremes when the pre-Robinson matrix is sorted to Robinson one, not all performed equally well.
- There were extreme differences in computing times for differrent objective functions.
- For $n \leq 100$, all objective functions performed well, except of Gar and $Rgar$ with $w = 0.25n$.
- For $n \geq 100$, Ber, although performing well, seems not to be practical to use due to enormous computing time.
- For $n = 500$, $Ware$ was the best performing function, measured by ρ, however, its computing time was enormous and the differrence in ρ, compared to the second best function, H, was minimal. Similarly $Wrcwg$ also performed well, but its computing time was almost 50 times longer than that of H.
- The best performing, thus recommended, functions for all n were H, $Psis$, and Lsq.
- There was almost no drop in performance in performance when a small amount of "noise" was introduced to the matrix, so that it was not sortable to the Robinson form. This result may be very important in practice.

Acknowledgments

The study was financially supported by the Institute of Geology of the Academy of Sciences of the Czech Republic, v.v.i. Internal Project #9349, RVO:6798583. Special thanks to Alexandros Kostopoulos and Yannis Goulermas, University of Liverpool, for their gracious help with

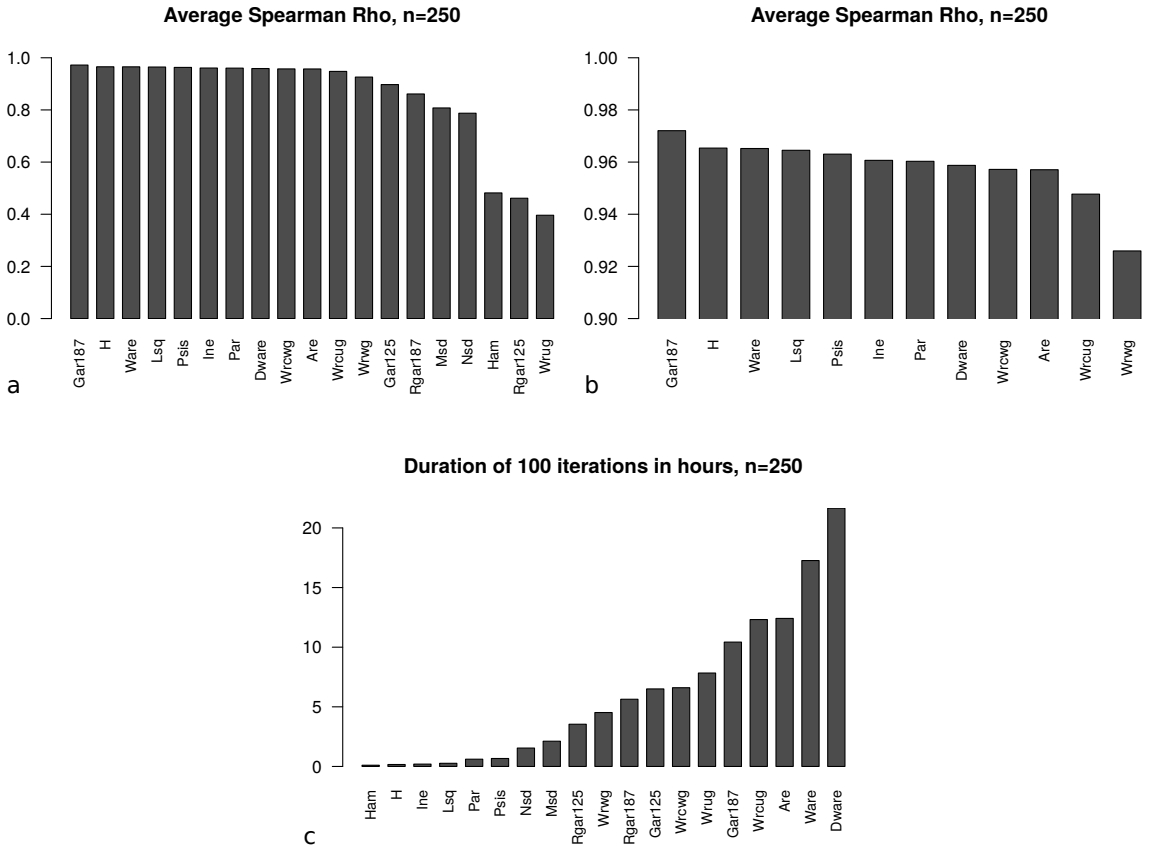

Figure 12: Performance of computationally feasible best objective functions (a, b), and duration of 100 iterations in hours (c) for 250×250 imperfect Robinson matrix.

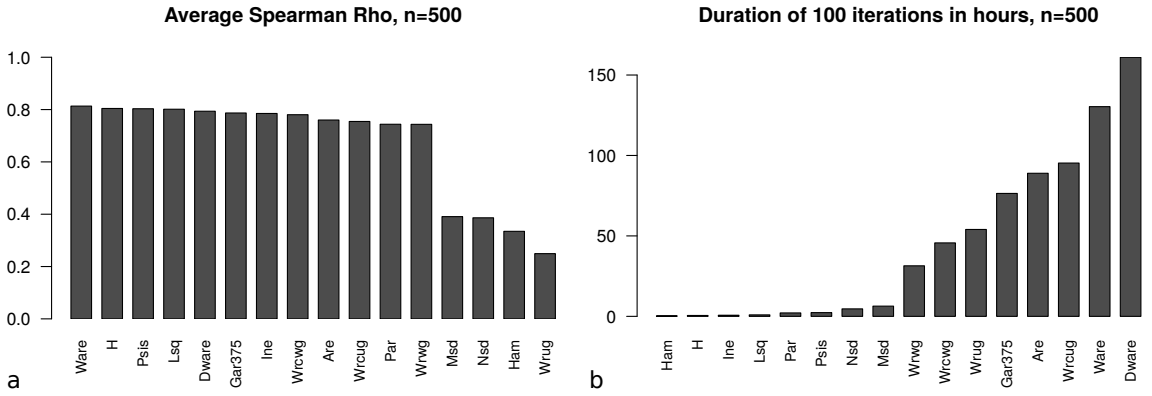

Figure 13: Performance of computationally feasible best objective functions (a), and duration of 100 iterations in hours (b) for 500×500 imperfect Robinson matrix.

debugging the code, Jindrich Hladil, Leona Chadimova, and Jana Frojdova, Institute of Geology of the Academy of Sciences of the Czech Republic, v.v.i., for proofreading of the text, and Miroslav Fridrich and Jaromir Havlica, Institute of Chemical Process Fundamentals of the ASCR, v.v.i., for computer time and precious help.

References

Ambati BK, Ambati J, Mokhtar MM (1991). "Heuristic Combinatorial Optimization by Simulated Darwinian Evolution: A Polynomial Time Algorithm for the Traveling Salesman Problem." *Biological Cybernetics*, **65**(1), 31–35.

Banzhaf W (1990). "The Molecular Traveling Salesman." *Biological Cybernetics*, **64**(1), 7–14.

Barthelemy JP, Brucker F (2001). "NP-hard Approximation Problems in Overlapping Clustering." *Journal of Classification*, **18**(2), 159–183.

Bertin J (1981). *Graphics and Graphic Information Processing*. Walter de Gruyter.

Brusco M, Stahl S (2000). "Using Quadratic Assignment Methods to Generate Initial Permutations for Least-Squares Unidimensional Scaling of Symmetric Proximity Matrices." *Journal of Classification*, **17**(2), 197–223.

Brusco MJ (2002). "Integer Programming Methods for Seriation and Unidemensional Scaling of Proximity Matrices: A Review and Some Extensions." *Journal of Classification*, **19**(1), 45–67.

Brusco MJ, Kohn HF, Stahl S (2008). "Heuristic Implementation of Dynamic Programming for Matrix Permutation Problems in Combinatorial Data Analysis." *Psychometrika*, **73**(3), 503–522.

Caraux G, Pinloche S (2005). "**PermutMatrix**: A Graphical Environment to Arrange Gene Expression Profiles in Optimal Linear Order." *Bioinformatics*, **21**(7), 1280–1281.

Cejchan PA (2014a). "Bibliography of seriation methods." URL http://www.gli.cas.cz/home/cejchan/db/SeriationBib.tar.

Cejchan PA (2014b). ***seriation**: Functions to Seriate Matrices to (Anti)Robinson Form*. URL https://code.google.com/p/seriation.

Cerny V (1985). "Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm." *Journal of Optimization Theory and Applications*, **45**, 41–51.

Chen CH (2002). "Generalized Association Plots: Information Visualization via Iteratively Generated Correlation Matrices." *Statistica Sinica*, **12**(1), 7–30.

Fogel DB (1988). "An Evolutionary Approach to the Traveling Salesman Problem." *Biological Cybernetics*, **60**(2), 139–144.

Garey MR, Johnson DS (1979). "Computers and Intractability: An Introduction to the Theory of NP-completeness."

Grefenstette JJ (1987). "Incorporating Problem Specific Knowledge into Genetic Algorithms." *Genetic Algorithms and Simulated Annealing*, **4**, 42–60.

Hahsler M, Hornik K, Buchta C (2008). "Getting Things in Order: An Introduction to the R Package **seriation**." *Journal of Statistical Software*, **25**(3), 1–34.

Holland JH (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press.

Hubert L, Arabie P, Meulman J (2001). *Combinatorial Data Analysis: Optimization by Dynamic Programming.* Society for Industrial Mathematics.

Hubert L, Schultz J (1976). "Quadratic Assignment as a General Data Analysis Strategy." *British Journal of Mathematical and Statistical Psychology*, **29**(2), 190–241.

Kirkpatrick S, Vecchi M, *et al.* (1983). "Optimization by Simmulated Annealing." *Science*, **220**(4598), 671–680.

Kostopoulos T, Goulermas J (2014). "A Hybrid Stochastic Algorithm for One- and Two-mode Seriation." *In preparation.*

Liiv I (2010). "Seriation and Matrix Reordering Methods: An Historical Overview." *Statistical Analysis and Data Mining*, **3**(2), 70–91.

Marcotorchino F (1987). "Block Seriation Problems: A Unified Approach." *Applied Stochastic Models and Data Analysis*, **3**, 73–91.

Merz P, Freisleben B (1997). "Genetic Local Search for the TSP: New Results." In *IEEE International Conference on Evolutionary Computation, 1997*, pp. 159–164.

Michalewicz Z (1992). *Genetic Algorithms + Data Structure = Evolution Programs.* Springer-Verlag.

Miklós I, Somodi I, Podani J (2005). "Rearrangement of Ecological Data Matrices via Markov Chain Monte Carlo Simulation." *Ecology*, **86**(12), 3398–3410.

Niermann S (2005). "Optimizing the Ordering of Tables with Evolutionary Computation." *The American Statistician*, **59**(1), 41–46.

Oliver I, Smith D, Holland J (1987). "A Study of Permutation Crossover Operators on the TSP, Genetic Algorithms and Their Applications." In *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 224–230.

Pilhofer A, Gribov A, Unwin A (2012). "Comparing Clusterings Using Bertin's Idea." *IEEE Transactions on Visualization and Computer Graphics*, **18**(12), 2506–2515.

Pirlot M (1996). "General local search methods." *European Journal of Operational Research*, **92**(3), 493–511.

Podani J, *et al.* (1994). *Multivariate Data Analysis in Ecology and Systematics: A Methodological Guide to the* ***SYN-TAX 5.0*** *Package.* SPB Academic Publishing.

Robinson WS (1951). "A Method for Chronologically Ordering Archaeological Deposits." *American Antiquity*, **16**, 293–301.

Sallabi OM, El-Haddad Y (2009). "An Improved Genetic Algorithm to Solve the Traveling Salesman Problem." *Proceedings of the World Academy of Science: Engineering & Technology*, **52**(3), 471–474.

Streng R (1991). "Classification and Seriation by Iterative Reordering of a Data Matrix." In *Classification, Data Analysis, and Knowledge Organization*, pp. 121–130. Springer-Verlag.

Streng R, Schönfelder P (1978). "Ein Heuristisches Computer-Programm zum Ordnung Pflanzensoziologischer Tabellen." *Hoppea*, **37**, 407–433.

Syswerda G (1991). "Schedule Optimization Using Genetic Algorithms." In *Handbook of Genetic Algorithms*, pp. 332–349. Van Nostrand Reinhold.

Szczotka F (1972). "On a Method of Ordering and Clustering of Objects." *Zastosowania Mathematyki*, **13**, 23–33.

Tien YJ, Lee YS, Wu HM, Chen CH (2008). "Methods for Simultaneously Identifying Coherent Local Clusters with Smooth Global Patterns in Gene Expression Profiles." *BMC Bioinformatics*, **9**(1), 155.

Van Laarhoven PJ, Aarts EH (1987). *Simulated Annealing.* Springer-Verlag.

Wu HM, Tien YJ, Chen CH (2010). "GAP: A Graphical Environment for Matrix Visualization and Cluster Analysis." *Computational Statistics & Data Analysis*, **54**(3), 767–778.

Affiliation:

Peter A. Cejchan
Laboratory of Paleobiology and Paleoecology
Institute of Geology
Academy of Sciences of the Czech Republic, v.v.i.
Rozvojova 269
165 00 Praha 6 Lysolaje, Czech Republic
E-mail: cej@gli.cas.cz
URL: http://www.gli.cas.cz/home/cejchan/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd