Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 890589, 15 pages
http://dx.doi.org/10.1155/2013/890589

Research Article

An Improved Exact Algorithm for Least-Squares Unidimensional Scaling

Gintaras Palubeckis

Faculty of Informatics, Kaunas University of Technology, Studentu 50-408, 51368 Kaunas, Lithuania

Correspondence should be addressed to Gintaras Palubeckis; gintaras@soften.ktu.lt

Received 16 October 2012; Accepted 31 March 2013

Academic Editor: Frank Werner

Given n objects and an $n \times n$ symmetric dissimilarity matrix D with zero main diagonal and nonnegative off-diagonal entries, the least-squares unidimensional scaling problem asks to find an arrangement of objects along a straight line such that the pairwise distances between them reflect dissimilarities represented by the matrix D. In this paper, we propose an improved branch-and-bound algorithm for solving this problem. The main ingredients of the algorithm include an innovative upper bounding technique relying on the linear assignment model and a dominance test which allows considerably reducing the redundancy in the enumeration process. An initial lower bound for the algorithm is provided by an iterated tabu search heuristic. To enhance the performance of this heuristic we develop an efficient method for exploring the pairwise interchange neighborhood of a solution in the search space. The basic principle and formulas of the method are also used in the implementation of the dominance test. We report computational results for both randomly generated and real-life based problem instances. In particular, we were able to solve to guaranteed optimality the problem defined by a 36×36 Morse code dissimilarity matrix.

1. Introduction

Least-squares (or L_2) unidimensional scaling is an important optimization problem in the field of combinatorial data analysis. It can be stated as follows. Suppose that there are n objects and, in addition, there are pairwise dissimilarity data available for them. These data form an $n \times n$ symmetric dissimilarity matrix, $D = (d_{ij})$, with zero main diagonal and nonnegative off-diagonal entries. The problem is to arrange the objects along a straight line such that the pairwise distances between them reflect dissimilarities given by the matrix D. Mathematically, the problem can be expressed as

$$\min \Phi(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(d_{ij} - \left|x_i - x_j\right|\right)^2, \tag{1}$$

where x denotes the vector $(x_1, \ldots, x_n)$ and x_i, $i \in \{1, \ldots, n\}$, is the coordinate for the object i on the line. We note that two (or even more) objects may have the same coordinate. In the literature, the double sum given by (1) is called a *stress function* [1].

Unidimensional scaling and its generalization, multidimensional scaling [1, 2], are dimensionality reduction techniques for exploratory data analysis and data mining [3]. The applicability of these techniques has been reported in a diversity of fields. In recent years, unidimensional scaling has found new applications in several domains including genetics, psychology, and medicine. Caraux and Pinloche [4] developed an environment to graphically explore gene expression data. This environment offers unidimensional scaling methods, along with some other popular techniques. Ng et al. [5] considered the problem of constructing linkage disequilibrium maps for human genome. They proposed a method for this problem which is centered on the use of a specific type of the unidimensional scaling model. Armstrong et al. [6] presented a study in which they evaluated interest profile differences across gender, racial-ethnic group, and employment status. The authors used two measures of profile similarity, the so-called Q correlation and Euclidean distances. In the former case, unidimensional scaling appeared to be a good model for interpreting the obtained results. Hubert and Steinley [7] applied unidimensional scaling to analyze proximity data on the United States Supreme Court

Justices. They have found that agreement among the justices was better represented by this approach than using a hierarchical classification method. Dahabiah et al. [8, 9] proposed a data mining technique based on a unidimensional scaling algorithm. They applied this technique on a digestive endoscope database consisting of a large number of pathologies. Ge et al. [10] developed a method for unsupervised classification of remote sensing images. Unidimensional scaling plays a pivotal role in this classifier. Bihn et al. [11] examined the diversity of ant assemblages in the Atlantic Forest of Southern Brazil. To visualize nestedness of ant genera distribution, a unidimensional scaling approach was adopted.

From (1), we see that in order to solve the unidimensional scaling problem (UDSP for short) we need to find not only a permutation of objects, but also their coordinates preserving the order of the objects as given by the permutation. Thus, at first glance, it might seem that the UDSP is a much more difficult problem than other linear arrangement problems. Fortunately, this is not the case. As shown by Defays [12], the UDSP can be reduced to the following combinatorial optimization problem:

$$\max_{p\in\Pi} F(p) = \sum_{k=1}^{n}\left(\sum_{l=1}^{k-1} d_{p(k)p(l)} - \sum_{l=k+1}^{n} d_{p(k)p(l)}\right)^2, \quad (2)$$

where Π is the set of all permutations of $W = \{1,\ldots,n\}$, that is, the set of all vectors $(p(1), p(2),\ldots, p(n))$ such that $p(k) \in W$, $k = 1,\ldots,n$, and $p(k) \neq p(l)$, $k = 1,\ldots,n-1$, $l = k+1,\ldots,n$. Letting S_i, $i \in W$, denote the sum $\sum_{j=1, j\neq i}^{n} d_{ij}$, we can write

$$\begin{aligned} F(p) &= \sum_{k=1}^{n}\left(2\sum_{l=1}^{k-1} d_{p(k)p(l)} - S_{p(k)}\right)^2 \\ &= \sum_{k=1}^{n}\left(2\sum_{l=k+1}^{n} d_{p(k)p(l)} - S_{p(k)}\right)^2. \end{aligned} \quad (3)$$

Thus, the UDSP is essentially a problem over all permutations of n objects. With an optimal solution p^* to (2) at hand, we can compute the optimal coordinates for the objects by setting $x_{p^*(1)} = 0$ and using the following relationship [13] for $k+1 = 2,\ldots,n$:

$$\begin{aligned} x_{p^*(k+1)} &= x_{p^*(k)} + \left(\sum_{l=k+1}^{n} d_{p^*(k)p^*(l)} - \sum_{l=1}^{k-1} d_{p^*(k)p^*(l)}\right. \\ &\quad \left. - \sum_{l=k+2}^{n} d_{p^*(k+1)p^*(l)} + \sum_{l=1}^{k} d_{p^*(k+1)p^*(l)}\right) n^{-1}. \end{aligned} \quad (4)$$

There is an important body of the literature on algorithms for solving the UDSP. Since the feasible solutions to the problem are linear arrangements of objects, a natural approach to the UDSP is to apply a general dynamic programming paradigm. An algorithm based on this paradigm was proposed by Hubert and Arabie [14]. However, as pointed out in the papers by Hubert et al. [15] and Brusco and Stahl [13], the dynamic programming algorithm requires very large memory for storing the partial solutions and, in fact, runs out of memory long before running out of time. As reported in these papers, the largest UDSP instances solved using dynamic programming were of sizes 25 and 26. An alternative approach for solving the UDSP is to use the branch-and-bound method. The first algorithm following this approach is that of Defays [12]. In his algorithm, during branching, the objects are assigned in an alternating fashion to either the leftmost free position or the rightmost free position in the n-permutation under construction. For example, at level 4 of the search tree, the four selected objects are $p(1)$, $p(n)$, $p(2)$, and $p(n-1)$, and there are no object assigned to positions $3,\ldots,n-2$ in the permutation p. To fathom partial solutions, the algorithm uses a symmetry test, an adjacency test, and a bound test. Some more comments on the Defays approach were provided in the paper by Brusco and Stahl [13]. The authors of that paper also proposed two improved branch-and-bound algorithms for unidimensional scaling. The first of them adopts the above-mentioned branching strategy borrowed from the Defays method. Meanwhile, in the second algorithm of Brusco and Stahl, an object chosen for branching is always assigned to the leftmost free position. Brusco and Stahl [13, 16] presented improved bounding procedures and an interchange test expanding the adjacency test used by Defays [12]. These components of their branch-and-bound algorithms are briefly sketched in Section 2 and Section 4, respectively. Brusco and Stahl [13] have reported computational experience for both randomly generated and empirical dissimilarity matrices. The results suggest that their branch-and-bound implementations are superior to the Defays algorithm. Moreover, the results demonstrate that these implementations are able to solve UDSP instances that are beyond the capabilities of the dynamic programming method.

There are also a number of heuristic algorithms for unidimensional scaling. These include the smoothing technique [17], the iterative quadratic assignment heuristic [15], and simulated annealing adaptations [18, 19]. A heuristic implementation of dynamic programming for escaping local optima has been presented by Brusco et al. [20]. Computational results for problem instances with up to 200 objects were reported in [18–20].

The focus of this paper is on developing a branch-and-bound algorithm for the UDSP. The primary intention is to solve exactly some larger problems than those solved by previous methods. The significant features of our algorithm are the use of an innovative upper bounding procedure relying on the linear assignment model and the enhancement of the interchange test from [13]. In the current paper, this enhancement is called a dominance test. A good initial solution and thus lower bound are obtained by running an iterated tabu search algorithm at the root node of the search tree. To make this algorithm more efficient, we provide a novel mechanism and underlying formulas for searching the pairwise interchange neighborhood of a solution in $O(n^2)$

time, which is an improvement over a standard $O(n^3)$-time approach from the literature. The same formulas are also used in the dominance test. We present the results of empirical evaluation of our branch-and-bound algorithm and compare it against the better of the two branch-and-bound algorithms of Brusco and Stahl [13]. The largest UDSP instance that we were able to solve with our algorithm was the problem defined by a 36×36 Morse code dissimilarity matrix.

The remainder of this paper is organized as follows. In Section 2, we present an upper bound on solutions of the UDSP. In Section 3, we outline an effective tabu search procedure for unidimensional scaling and propose a method for exploring the pairwise interchange neighborhood. Section 4 gives a description of the dominance test for pruning nodes in the search. This test is used in our branch-and-bound algorithm presented in Section 5. The results of computational experiments are reported in Section 6. Finally, Section 7 concludes the paper.

2. Upper Bounds

In this section, we first briefly recall the bounds used by Brusco and Stahl [13] in their second branch-and-bound algorithm. Then we present a new upper bounding mechanism for the UDSP. We discuss the technical details of the procedure for computing our bound. We choose a variation of this procedure which yields slightly weakened bounds but is quite fast.

Like in the second algorithm of Brusco and Stahl, in our implementation of the branch-and-bound method partial solutions are constructed by placing each new object in the leftmost free position. For a partial solution $p = (p(1), \ldots, p(r))$, $r \in \{1, \ldots, n-1\}$, we denote by $W(p)$ the set of the objects in p. Thus $W(p) = \{p(1), \ldots, p(r)\}$, and $\overline{W}(p) = W \setminus W(p)$ is the set of unselected objects. At the root node of the search tree, p is empty, and $\overline{W}(p)$ is equal to W. The first upper bound used by Brusco and Stahl, when adopted to p, takes the following form:

$$U_{\mathrm{BS}} := F_r(p) + \sum_{i \in \overline{W}(p)} S_i^2, \tag{5}$$

where $F_r(p)$ denotes the sum of the terms in (3) corresponding to assigned objects. Thus, $F_r(p) = \sum_{k=1}^{r} (S_{p(k)} - 2\sum_{l=1}^{k-1} d_{p(k)p(l)})^2$.

To describe the second bound, we define $\widehat{D} = (\widehat{d}_{ij})$ to be an $n \times (n-1)$ matrix obtained from D by sorting the nondiagonal entries of each row of D in nondecreasing order. Let $\eta_{ik} = (S_i - 2\sum_{j=1}^{k-1} \widehat{d}_{ij})^2$ and $\mu_k = \max_{1 \leqslant i \leqslant n} \eta_{ik}$ for $k = 1, \ldots, \lceil n/2 \rceil$. For other values of k, μ_k is obtained from the equality $\mu_k = \mu_{n-k+1}$, $k \in \{1, \ldots, n\}$. The second upper bound proposed in [13] is given by the following expression:

$$F_r(p) + \sum_{k=r+1}^{n} \mu_k. \tag{6}$$

The previous outlined bounds are computationally attractive but not strong enough when used to prune the search space. In order to get a tighter upper bound, we resort to the linear assignment model

$$\begin{aligned} \max \ & \sum_{i=1}^{n-r} \sum_{j=1}^{n-r} e_{ij} y_{ij} \\ \text{s.t.} \ & \sum_{j=1}^{n-r} y_{ij} = 1, \quad i = 1, \ldots, n-r, \\ & \sum_{i=1}^{n-r} y_{ij} = 1, \quad j = 1, \ldots, n-r, \\ & y_{ij} \in \{0, 1\}, \quad i, j = 1, \ldots, n-r, \end{aligned} \tag{7}$$

where e_{ij}, $i, j = 1, \ldots, n-r$ are entries of an $(n-r) \times (n-r)$ profit matrix E with rows and columns indexed by the objects in the set $\overline{W}(p)$. Next, we will construct the matrix E. For a permutation $p \in \Pi$, we denote by $f_{p(k)}$ (resp., $f'_{p(k)}$) the sum of dissimilarities between object $p(k)$ and all objects preceding (resp., following) $p(k)$ in p. Thus, $f_{p(k)} = \sum_{l=1}^{k-1} d_{p(k)p(l)}$, $f'_{p(k)} = \sum_{l=k+1}^{n} d_{p(k)p(l)}$. From (3), we have

$$F(p) = \sum_{k=1}^{n} \left(2 \max\left(f_{p(k)}, f'_{p(k)}\right) - S_{p(k)}\right)^2, \tag{8}$$

and $2\max(f_{p(k)}, f'_{p(k)}) \geqslant S_{p(k)} = f_{p(k)} + f'_{p(k)}$ for each $k = 1, \ldots, n$. Suppose now that $p = (p(1), \ldots, p(r))$ is a partial solution and $\Pi(p)$ is the set of all n-permutations that can be obtained by extending p. Assume, for the sake of simplicity, that $\overline{W}(p) = \{1, \ldots, n-r\}$. In order to construct a suitable matrix E, we have to bound $f_{p(k)}$ and $f'_{p(k)}$ in (8) from above. To this end, for any $i \in \overline{W}(p)$, let $(d'_{i1}, \ldots, d'_{i,n-r-1})$ denote the list obtained by sorting d_{ij}, $j \in \overline{W}(p) \setminus \{i\}$, in nonincreasing order. To get the required bounds, we make use of the sums $\gamma_{ij} = \sum_{l=1}^{j} d'_{i1}$, $i \in \overline{W}(p)$, $j = 0, 1, \ldots, n-r-1$, and $\rho_{ik} = \sum_{l=1}^{r} d_{ip(l)} + \gamma_{i,k-r-1}$, $i \in \overline{W}(p)$, $k = r+1, \ldots, n$. Suppose that $\widehat{p}$ is any permutation in $\Pi(p)$ and $\widehat{p}(k) = i$ for a position $k \in \{r+1, \ldots, n\}$. Then it is easy to see that $f_{\widehat{p}(k)} = f_i \leqslant \rho_{ik}$ and $f'_{\widehat{p}(k)} = f'_i \leqslant \gamma_{i,n-k}$. Indeed, in the case of $f_{\widehat{p}(k)}$, for example, we have $f_{\widehat{p}(k)} = \sum_{l=1}^{r} d_{ip(l)} + \sum_{l=r+1}^{k-1} d_{i\widehat{p}(l)}$, and the last term in this expression is less than or equal to $\gamma_{i,k-r-1}$. We define E to be the matrix with entries $e_{ij} = (2\max(\rho_{i,r+j}, \gamma_{i,n-r-j}) - S_i)^2$, $i, j = 1, \ldots, n-r$. Let U^* denote the optimal value of the linear assignment problem (LAP) (7) with the profit matrix E. From the definition of E, we immediately obtain an upper bound on the optimal value of the objective function F.

Proposition 1. *For a partial solution $p = (p(1), \ldots, p(r))$, $U := F_r(p) + U^* \geqslant \max_{\widehat{p} \in \Pi(p)} F(\widehat{p})$.*

From the definition of ρ and γ values, it can be observed that, with the increase of k, ρ_{ik} increases, while $\gamma_{i,n-k}$ decreases. Therefore, when constructing the ith row of the matrix E, it is expedient to first find the largest $k \in \{r+1, \ldots, n\}$ for which $\rho_{ik} \leqslant \gamma_{i,n-k}$. We denote such k by $\widetilde{k}$ and

assume that $\widetilde{k} = r$ if $\rho_{i,r+1} > \gamma_{i,n-r-1}$. Then $e_{i,k-r} = (2\gamma_{i,n-k} - S_i)^2$ for $k = r+1, \ldots, \widetilde{k}$, and $e_{i,k-r} = (2\rho_{ik} - S_i)^2$ for $k = \widetilde{k}+1, \ldots, n$.

The most time-consuming step in the computation of the upper bound U is solving the linear assignment problem. In order to provide a computationally less expensive bounding method we replace U^* with the value of a feasible solution to the dual of the LAP (7). To get the bound, we take $\alpha_i = \max_{1 \leqslant k \leqslant n-r} e_{ik} = e_{i,n-r} = S_i^2$, $i = 1, \ldots, n-r$, $\beta_j = \max_{1 \leqslant i \leqslant n-r}(e_{ij} - \alpha_i)$, $j = 1, \ldots, n-r$. Clearly, $\alpha_i + \beta_j \geqslant e_{ij}$ for each pair $i, j = 1, \ldots, n-r$. Thus the vector $(\alpha_1, \ldots, \alpha_{n-r}, \beta_1, \ldots, \beta_{n-r})$ satisfies the constraints of the dual of the LAP instance of interest. Let $\widetilde{U} = \sum_{i=1}^{n-r} \alpha_i + \sum_{j=1}^{n-r} \beta_j$. Then, from the duality theory of linear programming, we can establish the following upper bound for $F(\widehat{p})$.

Proposition 2. *For a partial solution* $p = (p(1), \ldots, p(r))$, $U' := F_r(p) + \widetilde{U} \geqslant \max_{\widehat{p} \in \Pi(p)} F(\widehat{p})$.

Proof. The inequality follows from Proposition 1 and the fact that $\widetilde{U} \geqslant U^*$. □

One point that needs to be emphasized is that U' strengthens the first bound proposed by Brusco and Stahl.

Proposition 3. $U' \leqslant U_{BS}$.

Proof. Indeed, $F_r(p) + \sum_{i \in \overline{W}(p)} S_i^2 = F_r(p) + \sum_{i=1}^{n-r} \alpha_i \geqslant F_r(p) + \widetilde{U}$. The equality in the previous chain of expressions comes from the choice of α_i and our assumption on $\overline{W}(p)$, whereas the inequality is due to the fact that $\beta_j \leqslant 0$ for each $j = 1, \ldots, n-r$. □

As an example, we consider the following 4×4 dissimilarity matrix borrowed from Defays [12]:

$$\begin{bmatrix} - & 4 & 2 & 3 \\ 4 & - & 7 & 2 \\ 2 & 7 & - & 4 \\ 3 & 2 & 4 & - \end{bmatrix}. \tag{9}$$

The same matrix was also used by Brusco and Stahl [13] to illustrate their bounding procedures. From Table 2 in their paper, we find that the solution $p = (2, 4, 1, 3)$ is optimal for this instance of the UDSP. The optimal value is equal to 388. We compute the proposed upper bound for the whole Defays problem. In this case, $\overline{W}(p) = \{1, \ldots, 4\}$ and $F_r(p) = 0$. The matrix E is constructed by noting that $\widetilde{k} = 2$ for each $i = 1, \ldots, 4$. It takes the following form:

$$\begin{bmatrix} 81 & 25 & 25 & 81 \\ 169 & 81 & 81 & 169 \\ 169 & 81 & 81 & 169 \\ 81 & 25 & 25 & 81 \end{bmatrix}. \tag{10}$$

Using the expressions for α_i and β_j, we find that $\alpha_1 = \alpha_4 = 81$, $\alpha_2 = \alpha_3 = 169$, $\beta_1 = \beta_4 = 0$, and $\beta_2 = \beta_3 = -56$. Thus, $\widetilde{U} = \sum_{i=1}^{4} \alpha_i + \sum_{j=1}^{4} \beta_j = 500 - 112 = 388 = \max_{p \in \Pi} F(p)$. Hence, our bound for the Defays instance is tight. For comparison, the bound (5) is equal to $\sum_{i=1}^{4} \alpha_i = 500$. To obtain the second bound of Brusco and Stahl, we first compute the μ values. Since $\mu_1 = \mu_4 = 169$ and $\mu_2 = \mu_3 = 81$, the bound (6) (taking the form $\sum_{i=1}^{4} \mu_i$) is also equal to 500.

3. Tabu Search

As it is well known that the effectiveness of the bound test in branch-and-bound algorithms depends on the value of the best solution found throughout the search process. A good strategy is to create a high-quality solution before the search is started. For this task, various heuristics can be applied. Our choice was to use the tabu search (TS) technique. It should be noted that it is not the purpose of this paper to investigate heuristic algorithms for solving the UDSP. Therefore, we restrict ourselves merely to presenting an outline of our implementation of tabu search.

For $p \in \Pi$, let p_{km} denote the permutation obtained from p by interchanging the objects in positions k and m. The set $N_2(p) = \{p_{km} \mid k, m = 1, \ldots, n, k < m\}$ is the 2-interchange neighborhood of p. At each iteration, the tabu search algorithm performs an exploration of the neighborhood $N_2(p)$. A subset of solutions in $N_2(p)$ are evaluated by computing $\Delta(p, k, m) = F(p_{km}) - F(p)$. In order to get better quality solutions we apply TS iteratively. This technique, called *iterated tabu search* (ITS), consists of the following steps:

(1) initialize the search with some permutation of the objects;

(2) apply a tabu search procedure. Check if the termination condition is satisfied. If so, then stop. Otherwise proceed to (3);

(3) apply a solution perturbation procedure. Return to (2).

In our implementation, the search is started from a randomly generated permutation. Also, some data are prepared to be used in the computation of the differences $\Delta(p, k, m)$. We will defer this issue until later in this section. In step (2) of ITS, at each iteration of the TS procedure, the neighborhood $N_2(p)$ of the current solution p is explored. The procedure selects a permutation $p_{km} \in N_2(p)$, which either improves the best available objective function value so far or, if this does not happen, has the largest value of $\Delta(p, i, j)$ among all permutations $p_{ij} \in N_2(p)$ for which the objects in positions i and j are not in the tabu list. The permutation p_{km} is then accepted as a new current solution. The number of iterations in the tabu search phase is bounded above by a parameter of the algorithm. The goal of the solution perturbation phase is to generate a permutation which is employed as the starting point for the next invocation of the tabu search procedure. This is achieved by taking the current permutation and performing a sequence of pairwise interchanges of objects. Throughout this process, each object is allowed to change its position in the permutation at most once. At each step, first a set of feasible pairs of permutation positions with the largest Δ values is formed. Then a pair, say (k, m), is chosen randomly from this set, and objects in positions k and m are switched. We omit some of the details of

the algorithm. Similar implementations of the ITS technique for other optimization problems are thoroughly described, for example, in [21, 22]. We should note that also other enhancements of the generic tabu search strategy could be used for our purposes. These include, for example, multipath adaptive tabu search [23] and bacterial foraging tabu search [24] algorithms.

In the rest of this section, we concentrate on how to efficiently compute the differences $\Delta(p,k,m)$. Certainly, this issue is central to the performance of the tabu search procedure. But more importantly, evaluation of pairwise interchanges of objects stands at the heart of the dominance test, which is one of the crucial components of our algorithm. As we will see in the next section, the computation of the Δ values is an operation the dominance test is based on.

Brusco [19] derived the following formula to calculate the difference between $F(p_{km})$ and $F(p)$:

$$\begin{aligned}
&\Delta(p,k,m)\\
&= \left(4\sum_{l=k+1}^{m} d_{p(k)p(l)}\right)\left(\left(\sum_{l=k+1}^{m} d_{p(k)p(l)}\right) + 2f_{p(k)} - S_{p(k)}\right)\\
&\quad + \left(4\sum_{l=k}^{m-1} d_{p(l)p(m)}\right)\left(\left(\sum_{l=k}^{m-1} d_{p(l)p(m)}\right) - 2f_{p(m)} + S_{p(m)}\right)\\
&\quad + \sum_{l=k+1}^{m-1} 4\left(d_{p(l)p(m)} - d_{p(k)p(l)}\right)\left(\left(d_{p(l)p(m)} - d_{p(k)p(l)}\right)\right.\\
&\qquad \left. + 2f_{p(l)} - S_{p(l)}\right).
\end{aligned} \tag{11}$$

Using (11), $\Delta(p,k,m)$ for all permutations in $N_2(p)$ can be computed in $O(n^3)$ time. However, this approach is actually too slow to be efficient. In order to describe a more elaborate approach we introduce three auxiliary $n \times n$ matrices $A = (a_{iq})$, $B = (b_{iq})$, and $C = (c_{ij})$ defined for a permutation $p \in \Pi$. Let us consider an object $i = p(s)$. The entries of the ith row of the matrices A and B are defined as follows: if $q < s$, then $a_{iq} = \sum_{l=q}^{s-1} d_{ip(l)}$, $b_{iq} = \sum_{l=q}^{s-1} d_{ip(l)}(d_{ip(l)} + 2f_{p(l)} - S_{p(l)})$; if $q > s$, then $a_{iq} = \sum_{l=s+1}^{q} d_{ip(l)}$, $b_{iq} = \sum_{l=s+1}^{q} d_{ip(l)}(-d_{ip(l)} + 2f_{p(l)} - S_{p(l)})$; if $q = s$, then $a_{iq} = b_{iq} = 0$. Turning now to the matrix C, we suppose that $j = p(t)$. Then the entries of C are given by the following formula:

$$c_{ij} = \begin{cases} \displaystyle\sum_{l=\min(s,t)+1}^{\max(s,t)-1} d_{ip(l)}d_{jp(l)} & \text{if } |s-t| > 1\\ 0 & \text{if } |s-t| = 1. \end{cases} \tag{12}$$

Unlike A and B, the matrix C is symmetric. We use the matrices A, B, and C to cast (11) in a form more suitable for both the TS heuristic and the dominance test.

Proposition 4. *For $p \in \Pi$, $k \in \{1,\ldots,n-1\}$, and $m \in \{k+1,\ldots,n\}$,*

$$\begin{aligned}
\Delta(p,k,m) &= 4a_{p(k)m}\left(a_{p(k)m} + 2a_{p(k),1} - S_{p(k)}\right)\\
&\quad + 4a_{p(m)k}\left(a_{p(m)k} - 2a_{p(m),1} + S_{p(m)}\right)\\
&\quad + 4\left(b_{p(m),k+1} - b_{p(k),m-1} - 2c_{p(k)p(m)}\right).
\end{aligned} \tag{13}$$

Proof. First, notice that the sums $\sum_{l=k+1}^{m} d_{p(k)p(l)}$ and $\sum_{l=k}^{m-1} d_{p(l)p(m)}$ in (11) are equal, respectively, to $a_{p(k)m}$ and $a_{p(m)k}$. Next, it can be seen that $f_{p(k)} = a_{p(k),1}$ and $f_{p(m)} = a_{p(m),1}$. Finally, the third term in (11), ignoring the factor of 4, can be rewritten as

$$\begin{aligned}
&\sum_{l=k+1}^{m-1} d_{p(l)p(m)}\left(d_{p(l)p(m)} + 2f_{p(l)} - S_{p(l)}\right)\\
&\quad - \sum_{l=k+1}^{m-1} d_{p(l)p(m)}d_{p(k)p(l)} - \sum_{l=k+1}^{m-1} d_{p(k)p(l)}d_{p(l)p(m)}\\
&\quad - \sum_{l=k+1}^{m-1} d_{p(k)p(l)}\left(-d_{p(k)p(l)} + 2f_{p(l)} - S_{p(l)}\right).
\end{aligned} \tag{14}$$

It is easy to see that the first sum in (14) is equal to $b_{p(m),k+1}$, each of the next two sums is equal to $c_{p(k)p(m)}$, and the last sum is equal to $b_{p(k),m-1}$. After simple manipulations, we arrive at (13). □

Having the matrices $A, B,$ and C and using (13), the 2-interchange neighborhood of p can be explored in $O(n^2)$ time. The question that remains to be addressed is how efficiently the entries of these matrices can be updated after the replacement of p by a permutation p' chosen from the neighborhood $N_2(p)$. Suppose that p' is obtained from p by interchanging the objects in positions k and m. Then, we can write the following formulas that are involved in updating the matrices A and B:

$$a_{iq} := a_{i,q-1} + d_{ip'(q)}, \tag{15}$$

$$a_{iq} := a_{i,q+1} + d_{ip'(q)}, \tag{16}$$

$$b_{iq} := b_{i,q-1} + d_{ip'(q)}\left(2a_{p'(q),1} - S_{p'(q)} - d_{ip'(q)}\right), \tag{17}$$

$$b_{iq} := b_{i,q+1} + d_{ip'(q)}\left(2a_{p'(q),1} - S_{p'(q)} + d_{ip'(q)}\right). \tag{18}$$

From now on, we will assume that the positions s and t of the objects i and j are defined with respect to the permutation p'. More formally, we have $i = p'(s)$, $j = p'(t)$. The procedure for updating A, B, and C is summarized in the following:

(1) for each object $i \in \{1,\ldots,n\}$ do the following:

(1.1) if $s < k$, then compute a_{iq}, $q = k,\ldots,m-1$, by (15) and b_{iq}, $q = k,\ldots,n$, by (17);

(1.2) if $s > m$, then compute a_{iq}, $q = m, m-1,\ldots,k+1$, by (16) and b_{iq}, $q = m, m-1,\ldots,1$, by (18);

(1.3) if $k < s < m$, then compute a_{iq} by (15) for $q = m,\ldots,n$ and by (16) for $q = k, k-1,\ldots,1$;

(1.4) if $s = k$ or $s = m$, then first set $a_{is} := 0$ and afterwards compute a_{iq} by (15) for $q = s + 1, \ldots, n$ and by (16) for $q = s - 1, s - 2, \ldots, 1$;

(1.5) if $k \leqslant s \leqslant m$, then first set $b_{is} := 0$ and afterwards compute b_{iq} by (17) for $q = s+1, \ldots, n$ and by (18) for $q = s - 1, s - 2, \ldots, 1$;

(2) for each pair of objects $i, j \in \{1, \ldots, n\}$ such that $s < t$ do the following:

(2.1) if $s < k < t < m$, then set $c_{ij} := c_{ij} + d_{ip'(k)}d_{jp'(k)} - d_{ip'(m)}d_{jp'(m)}$;

(2.2) if $k < s < m < t$, then set $c_{ij} := c_{ij} + d_{ip'(m)}d_{jp'(m)} - d_{ip'(k)}d_{jp'(k)}$;

(2.3) if $|\{s, t\} \cap \{k, m\}| = 1$, then compute c_{ij} anew using (12);

(2.4) if c_{ij} has been updated in (2.1)–(2.3), then set $c_{ji} := c_{ij}$.

It can be noted that for some pairs of objects i and j the entry c_{ij} remains unchanged. There are five such cases: (1) $s, t < k$; (2) $k < s, t < m$; (3) $s, t > m$; (4) $s < k < m < t$; (5) $\{s, t\} = \{k, m\}$.

We now evaluate the time complexity of the described procedure. Each of Steps (1.1) to (1.5) requires $O(n)$ time. Thus, the complexity of Step (1) is $O(n^2)$. The number of operations in Steps (2.1), (2.2), and (2.4) is $O(1)$, and that in Step (2.3) is $O(n)$. However, the latter is executed only $O(n)$ times. Therefore, the overall complexity of Step (2), and hence of the entire procedure, is $O(n^2)$.

Formula (13) and the previous described procedure for updating matrices A, B, and C lie at the core of our implementation of the TS method. The matrices A, B, and C can be initialized by applying the formulas (15)–(18) and (12) with respect to the starting permutation. The time complexity of initialization is $O(n^2)$ for A and B, and $O(n^3)$ for C. The auxiliary matrices A, B, and C and the expression (13) for $\Delta(p, k, m)$ are also used in the dominance test presented in the next section. This test plays a major role in pruning the search tree during the branching and bounding process.

4. Dominance Test

As is well known, the use of dominance tests in enumerative methods allows reducing the search space of the optimization problem of interest. A comprehensive review of dominance tests (or rules) in combinatorial optimization can be found in the paper by Jouglet and Carlier [25]. For the problem we are dealing with in this study, Brusco and Stahl [13] proposed a dominance test for discarding partial solutions, which relies on performing pairwise interchanges of objects. To formulate it, let us assume that $p = (p(1), \ldots, p(r))$, $r < n$, is a partial solution to the UDSP. The test checks whether $\Delta(p, k, r) > 0$ for at least one of the positions $k \in \{1, \ldots, r - 1\}$. If this condition is satisfied, then we say that the test fails. In this case, p is thrown away. Otherwise, p needs to be extended by placing an unassigned object in position $r + 1$. The first condition in our dominance test is the same as in the test suggested by Brusco and Stahl. To compute $\Delta(p, k, r)$, we use subsets of entries of the auxiliary matrices A, B, and C defined in the previous section. Since only the sign of $\Delta(p, k, r)$ matters, the factor of 4 can be ignored in (13), and thus $\Delta'(p, k, r) = \Delta(p, k, r)/4$ instead of $\Delta(p, k, r)$ can be considered. From (13) we obtain

$$\begin{aligned}\Delta'(p, k, r) = {} & a_{p(k)r}\left(a_{p(k)r} + 2a_{p(k),1} - S_{p(k)}\right) \\ & + a_{p(r)k}\left(a_{p(r)k} - 2a_{p(r),1} + S_{p(r)}\right) \\ & + b_{p(r),k+1} - b_{p(k),r-1} - 2c_{p(k)p(r)}.\end{aligned} \tag{19}$$

We also take advantage of the case where $\Delta'(p, k, r) = 0$. We use the lexicographic rule for breaking ties. The test outputs a "Fail" verdict if the equality $\Delta'(p, k, r) = 0$ is found to hold true for some $k \in \{1, \ldots, r - 1\}$ such that $p(k) > p(r)$. Indeed, $p = (p(1), \ldots, p(k - 1), p(k), p(k + 1), \ldots, p(r))$ can be discarded because both p and partial solution $p' = (p(1), \ldots, p(k-1), p(r), p(k+1), \ldots, p(k))$ are equally good, but p is lexicographically larger than p'.

We note that the entries of the matrices A, B, and C used in (19) can be easily obtained from a subset of entries of A, B, and C computed earlier in the search process. Consider, for example, $c_{p(k)p(r)}$ in (19). It can be assumed that $k \in \{1, \ldots, r - 2\}$ because if $k = r - 1$, then $c_{p(k)p(r)} = 0$. For simplicity, let $p(r) = i$ and $p(r - 1) = j$. To avoid ambiguity, let us denote by C' the matrix C that is obtained during processing of the partial solution $(p(1), \ldots, p(r - 2))$. The entry $c'_{p(k)i}$ of C' is calculated by (temporarily) placing the object i in position $r - 1$. Now, referring to the definition of C, it is easy to see that $c_{p(k)i} = c'_{p(k)i} + d_{p(k)j}d_{ij}$. Similarly, using formulas of types (15) and (17), we can obtain the required entries of the matrices A and B. Thus (19) can be computed in constant time. The complexity of computing $\Delta'(p, k, r)$ for all k is $O(n)$. The dominance test used in [13] relies on (11) and has complexity $O(n^2)$. So our implementation of the dominance test is significantly more time-efficient than that proposed by Brusco and Stahl.

In order to make the test even more efficient we, in addition, decided to evaluate those partial solutions that can be obtained from p by relocating the object $p(r)$ from position r to position $k \in \{1, \ldots, r - 2\}$ (here position $k = r - 1$ is excluded since in this case relocating would become a pairwise interchange of objects). Thus, the test compares p against partial solutions $p_k = (p(1), \ldots, p(k - 1), p(r), p(k), \ldots, p(r - 1))$, $k = 1, \ldots, r - 2$. Now suppose that p and p_k are extended to n-permutations $\tilde{p} \in \Pi$ and, respectively, $\tilde{p}_k \in \Pi$ such that, for each $l \in \{r + 1, \ldots, n\}$, the object in position l of $\tilde{p}$ coincides with that in position l of $\tilde{p}_k$. Let us define δ_{rk} to be the difference between $F(\tilde{p}_k)$ and $F(\tilde{p})$. This difference can be computed as follows:

$$\begin{aligned}\delta_{rk} = {} & \left(4\sum_{l=k}^{r-1} d_{p(r)p(l)}\right)\left(\left(\sum_{l=k}^{r-1} d_{p(r)p(l)}\right) - 2f_{p(r)} + S_{p(r)}\right) \\ & + 4\sum_{l=k}^{r-1} d_{p(r)p(l)}\left(d_{p(r)p(l)} + 2f_{p(l)} - S_{p(l)}\right).\end{aligned} \tag{20}$$

The previous expression is similar to the formula (6) in [19]. For completeness, we give its derivation in the Appendix. We can rewrite (20) using the matrices A and B.

Proposition 5. *For* $p \in \Pi$, $k \in \{1, \ldots, n-1\}$, *and* $r \in \{k+1, \ldots, n\}$,

$$\delta_{rk} = 4a_{p(r)k}\left(a_{p(r)k} - 2a_{p(r),1} + S_{p(r)}\right) + 4b_{p(r)k}. \quad (21)$$

The partial solution p is dominated by p_k if $\delta_{rk} > 0$. Upon establishing this fact for some $k \in \{1, \ldots, r-2\}$, the test immediately stops with the rejection of p. Of course, like in the case of (13), a factor of 4 in (21) can be dropped. The computational overhead of including the condition $\delta_{rk} > 0$ in the test is minimal because all the quantities appearing on the right-hand side of (21), except $b_{p(r),1}$, are also used in the calculation of $\Delta'(p, k, r)$ (see (19)). Since the loop over k needs to be executed the time complexity of this part of the test is $O(n)$. Certainly, evaluation of relocating the last added object in p gives more power to the test. Thus our dominance test is an improvement over the test described in the Brusco and Stahl [13] paper.

5. A Branch-and-Bound Algorithm

Our approach to the UDSP is to start with a good initial solution provided by the iterated tabu search heuristic invoked in the initialization phase of the algorithm and then proceed with the main phase where, using the branch-and-bound technique, a search tree is built. For the purpose of pruning branches of the search tree, the algorithm involves three tests: (1) the upper bound test (based on U'); (2) the dominance test; (3) the symmetry test. The first two of them have already been described in the prior sections. The last one is the most simple and cheap to apply. It hinges on the fact that the values of the objective function F at a permutation $p \in \Pi$ and its reverse $\widetilde{p}$ defined by $\widetilde{p}(i) = p(n-i+1)$, $i = 1, \ldots, n$, are the same. Therefore, it is very natural to demand that some fixed object in W, say w, being among those objects that are assigned to the first half of the positions in p. More formally, the condition is $i \leqslant \lceil n/2 \rceil$ where i is such that $p(i) = w$. If this condition is violated for $p = (p(1), \ldots, p(\lceil n/2 \rceil))$, then the symmetry test fails, and p is discarded. Because of the requirement to fulfill the previous condition, some care should be taken in applying the dominance test. In the case of $\Delta'(p, k, r) = 0$ and $p(k) > p(r)$ (see the paragraph following (19)), this test in our implementation of the algorithm gives a "Fail" verdict only if either $r \leqslant \lceil n/2 \rceil$ or $p(k) \neq w$.

Through the preliminary computational experiments, we were convinced that the upper bound test was quite weak when applied to smaller partial solutions. In other words, the time overhead for computing linear assignment-based bounds did not pay off for partial permutations $p = (p(1), \ldots, p(r))$ of length r that is small or modest compared to n. Therefore, it is reasonable to compute the upper bound U' (described in Section 2) only when $r \geqslant \lambda(D)$, where $\lambda(D)$ is a threshold value depending on the UDSP instance matrix D and thus also on n. The problem we face is how to determine an appropriate value for $\lambda(D)$. We combat this problem with a simple procedure consisting of two steps. In the first step, it randomly generates a relatively small number of partial permutations $p_1, \ldots, p_\tau$, each of length $\lfloor n/2 \rfloor$. For each permutation in this sample, the procedure computes the upper bound U'. Let $U'_1, \ldots, U'_\tau$ be the values obtained. We are interested in the normalized differences between these values and $F(p_{\text{heur}})$, where p_{heur} is the solution delivered by the ITS algorithm. So the procedure calculates the sum $Z = \sum_{i=1}^{\tau} 100(U'_i - F(p_{\text{heur}}))/F(p_{\text{heur}})$ and the average $R = Z/\tau$ of such differences. In the second step, using R, it determines the threshold value. This is done in the following way: if $R < R_1$, then $\lambda(D) = \lfloor n/3 \rfloor$; if $R_1 \leqslant R < R_2$, then $\lambda(D) = \lfloor n/2 \rfloor$; if $R > R_2$, then $\lambda(D) = \lfloor 3n/4 \rfloor$. The described procedure is executed at the initialization step of the algorithm. Its parameters are R_1, R_2, and the sample size τ.

Armed with tools for pruning the search space, we can construct a branch-and-bound algorithm for solving the UDSP. In order to make the description of the algorithm more readable we divide it into two parts, namely, the main procedure B&B (branch-and-bound) and an auxiliary procedure SelectObject. The former builds a search tree with nodes representing partial solutions. The task of SelectObject is either to choose an object, which can be used to extend the current permutation, or, if the current node becomes fathomed, to perform a backtracking step. In the description of the algorithm given in the following, the set of objects that can be used to create new branches is denoted by V. The main procedure goes as follows.

B&B

(1) Apply the ITS algorithm to get an initial solution p^* to a given instance of the UDSP.

Set $F^* := F(p^*)$.

(2) Calculate $\lambda(D)$.

(3) Initialize the search tree by creating the root node with the empty set V attached to it.

Set $L := 0$ (L denotes the tree level the currently considered node belongs to).

(4) Set $r := L$.

(5) Call SelectObject(r). It returns L and, if $L = r$, additionally some unassigned object (let it be denoted by v). If $L < 0$, then go to (7). Otherwise, check whether $L < r$. If so, then return to (4) (perform a backtracking step). If not (in which case $L = r$), then proceed to (6).

(6) Create a new node with empty V. Set $p(r+1) := v$. Increment L by 1, and go to (4).

(7) Calculate the coordinates for the objects by applying formula (4) with respect to the permutation p^* and stop.

Steps (1) and (2) of B&B comprise the initialization phase of the algorithm. The initial solution to the problem is produced using ITS. This solution is saved as the best solution found so far, denoted by p^*. If later a better solution is found, p^* is updated. This is done within the SelectObject

procedure. In the final step, B&B uses p^* to calculate the optimal coordinates for the objects. The branch-and-bound phase starts at Step (3) of B&B. Each node of the search tree is assigned a set V. The partial solution corresponding to the current node is denoted by p. The objects of V are used to extend p. Upon creation of a node (Step (3) for root and Step (6) for other nodes), the set V is empty. In SelectObject, V is filled with objects that are promising candidates to perform the branching operation on the corresponding node. SelectObject also decides which action has to be taken next. If $L = r$, then the action is "branch," whereas if $L < r$, the action is "backtrack." Step (4) helps to discriminate between these two cases. If $L = r$, then a new node in the next level of the search tree is generated. At that point, the partial permutation p is extended by assigning the selected object v to the position $r + 1$ in p (see Step (6)). If, however, $L < r$ and r is positive, then backtracking is performed by repeating Steps (4) and (5). We thus see that the search process is controlled by the procedure SelectObject. This procedure can be stated as follows.

SelectObject(r)

(1) Check whether the set V attached to the current node is empty. If so, then go to (2). If not, then consider its subset $V' = \{i \in V \mid i \text{ is unmarked and } u_i(p) > F^*\}$. If V' is nonempty, then mark an arbitrary object $v \in V'$, and return with it as well as with $L = r$. Otherwise go to (6).

(2) If $r > 0$, then go to (3). Otherwise, set $V := \{1, \ldots, n\}$, $u_i(p) := \infty$, $i = 1, \ldots, n$, and go to (4).

(3) If $r = n - 1$, then go to (5). Otherwise, for each $i \in \overline{W}(p)$, perform the following operations. Temporarily set $p(r + 1) := i$. Apply the symmetry test, the dominance test, and, if $r \geqslant \lambda(D)$, also the bound test to the resulting partial solution $p' = (p(1), \ldots, p(r + 1))$. If it passes all the tests, then do the following. Include i into V. If $r \geqslant \lambda(D)$, assign to $u_i(p)$ the value of the upper bound U' computed for the subproblem defined by the partial solution p'. If, however, $r < \lambda(D)$, then set $u_i(p) := \infty$. If after processing all objects in $\overline{W}(p)$, the set V remains empty, then go to (6). Otherwise proceed to (4).

(4) Mark an arbitrary object $v \in V$, and return with it as well as with $L = r$.

(5) Let $\overline{W}(p) = \{j\}$. Calculate the value of the solution $p = (p(1), \ldots, p(n - 1), p(n) = j)$. If $F(p) > F^*$, then set $p^* := p$ and $F^* := F(p)$.

(6) If $r > 0$, then drop the rth element of p. Return with $L := r - 1$.

In the previous description, $\overline{W}(p)$ denotes the set of unassigned objects, as before. The objects included in $V \subseteq \overline{W}(p)$ are considered as candidates to be assigned to the leftmost available position in p. At the root node, $V = \overline{W}(p)$. In all other cases, except when $|\overline{W}(p)| = 1$, an object becomes a member of the set V only if it passes two or, for longer p, three tests. The third of them, the upper bound test, amounts to check if $U' > F^*$. This condition is checked again on the second and subsequent calls to SelectObject for the same node of the search tree. This is done in Step (1) for each unmarked object $i \in V$. Such a strategy is reasonable because during the branch-and-bound process the value of F^* may increase, and the previously satisfied constraint $U' > F^*$ may become violated for some $i \in V$. To be able to identify such objects, the algorithm stores the computed upper bound U' as $u_i(p)$ at Step (3) of SelectObject.

A closer look at the procedure SelectObject shows that two scenarios are possible: (1) when this procedure is invoked following the creation of a new node of the tree (which is done either in Step (3) or in Step (6) of B&B); (2) when a call to the procedure is made after a backtracking step. The set V initially is empty, and therefore Steps (2) to (5) of SelectObject become active only in the first of these two scenarios. Depending on the current level of the tree, r, one of the three cases is considered. If the current node is the root, that is, $r = 0$, then Step (2) is executed. Since in this case the dominance test is not applicable, the algorithm uses the set V comprising all the objects under consideration. An arbitrary object selected in Step (4) is then used to create the first branch emanating from the root of the tree. If $r \in [1, \ldots, n - 2]$, then the algorithm proceeds to Step (3). The task of this step is to evaluate all partial solutions that can be obtained from the current partial permutation p by assigning an object from $\overline{W}(p)$ to the $(r + 1)$ th position in p. In order not to lose at least one optimal solution it is sufficient to keep only those extensions of p for which all the tests were successful. Such partial solutions are memorized by the set V. One of them is chosen in Step (4) to be used (in Step (6) of B&B) to generate the first son of the current node. If, however, no promising partial solution has been found and thus the set V is empty, then the current node can be discarded, and the search continued from its parent node. Finally, if $r = n - 1$, then Step (5) is executed in which the permutation p is completed by placing the remaining unassigned object in the last position of p. Its value is compared with that of the best solution found so far. The node corresponding to p is a leaf and, therefore, is fathomed. In the second of the above-mentioned two scenarios (Step (1) of the procedure), the already existing nonempty set V is scanned. The previously selected objects in V have been marked. They are ignored. For each remaining object i, it is checked whether $u_i(p) > F^*$. If so, then i is included in the set V'. If the resulting set V' is empty, then the current node of the tree becomes fathomed, and the procedure goes to Step (6). Otherwise, the procedure chooses a branching object and passes it to its caller B&B. Our implementation of the algorithm adopts a very simple enumeration strategy: branch upon an arbitrary object in V' (or V when Step (4) is performed).

It is useful to note that the tests in Step (3) are applied in increasing order of time complexity. The symmetry test is the cheapest of them, whereas the upper bound test is the most expensive to perform but still effective.

6. Computational Results

The main goal of our numerical experiments was to provide some measure of the performance of the developed

B&B algorithm as well as to demonstrate the benefits of using the new upper bound and efficient dominance test throughout the solution process. For comparison purposes, we also implemented the better of the two branch-and-bound algorithms of Brusco and Stahl [13]. In their study, this algorithm is denoted as BB3. When referring to it in this paper, we will use the same name. In order to make the comparison more fair, we decided to start BB3 by invoking the ITS heuristic. Thus, the initial solution from which the branch-and-bound phase of each of the algorithms starts is expected to be of the same quality.

6.1. Experimental Setup. Both our algorithm and BB3 have been coded in the C programming language, and all the tests have been carried out on a PC with an Intel Core 2 Duo CPU running at 3.0 GHz. As a testbed for the algorithms, three empirical dissimilarity matrices from the literature in addition to a number of randomly generated ones were considered. The random matrices vary in size from 20 to 30 objects. All off-diagonal entries are drawn uniformly at random from the interval $[1, 100]$. The matrices, of course, are symmetric.

We also used three dissimilarity matrices constructed from empirical data found in the literature. The first matrix is based on Morse code confusion data collected by Rothkopf [26]. In his experiment, the subjects were presented pairs of Morse code signals and asked to respond whether they thought the signals were the same or different. These answers were translated to the entries of the similarity matrix with rows and columns corresponding to the 36 alphanumeric characters (26 letters in the Latin alphabet and 10 digits in the decimal number system). We should mention that we found small discrepancies between Morse code similarity matrices given in various publications and electronic media. We have decided to take the matrix from [27], where it is presented with a reference to Shepard [28]. Let this matrix be denoted by $M = (m_{ij})$. Its entries are integers from the interval $[0, 100]$. Like in the study of Brusco and Stahl [13], the corresponding symmetric dissimilarity matrix $D = (d_{ij})$ was obtained by using the formula $d_{ij} = 200 - m_{ij} - m_{ji}$ for all off-diagonal entries and setting the diagonal of D to zero. We considered two UDSP instances constructed using Morse code confusion data—one defined by the 26×26 submatrix of D with rows and columns labeled by the letters and another defined by the full matrix D.

The second empirical matrix was taken from Groenen and Heiser [29]. We denote it by $G = (g_{ij})$. Its entry g_{ij} gives the number of citations in journal i to journal j. Thus the rows of G correspond to citing journals, and the columns correspond to cited journals. The dissimilarity matrix D was derived from G in two steps, like in [13]. First, a symmetric similarity matrix, $H = (h_{ij})$, was calculated by using the formulas $h_{ij} = h_{ji} = (g_{ij}/\sum_{l=1,l\neq i}^{n} g_{il}) + (g_{ji}/\sum_{l=1,l\neq j}^{n} g_{jl})$, $i, j = 1, \ldots, n$, $i < j$, and $h_{ii} = 0$, $i = 1, \ldots, n$. Then D was constructed with entries $d_{ij} = \lfloor 100(\max_{l,k} h_{lk} - h_{ij}) \rfloor$, $i, j = 1, \ldots, n$, $i \neq j$, and $d_{ii} = 0$, $i = 1, \ldots, n$.

The third empirical matrix used in our experiments was taken from Hubert et al. [30]. Its entries represent the dissimilarities between pairs of food items. As mentioned by Hubert et al. [30], this 45×45 food-item matrix, D_{food}, was constructed based on data provided by Ross and Murphy [31]. For our computational tests, we considered submatrices of D_{food} defined by the first $n \in \{20, 21, \ldots, 35\}$ food items. All the dissimilarity matrices we just described as well as the source code of our algorithm are publicly available at http://www.proin.ktu.lt/~gintaras/uds.html.

Based on the results of preliminary experiments with the B&B algorithm, we have fixed the values of its parameters: $\tau = 10$, $R_1 = 0$, and $R_2 = 5$. In the symmetry test, w was fixed at 1. The cutoff time for ITS was 1 second.

6.2. Results. The first experiment was conducted on a series of randomly generated full density dissimilarity matrices. The results of B&B and BB3 for them are summarized in Table 1. The dimension of the matrix D is encoded in the instance name displayed in the first column. The second column of the table shows, for each instance, the optimal value of the objective function F^* and, in parentheses, two related metrics. The first of them is the raw value of the stress function $\Phi(x)$. The second number in parentheses is the normalized value of the stress function that is calculated as $\Phi(x)/\sum_{i<j} d_{ij}$. The next two columns list the results of our algorithm: the number of nodes in the search tree and the CPU time reported in the form *hours* : *minutes* : *seconds* or *minutes* : *seconds* or *seconds*. The last two columns give these measures for BB3.

As it can be seen from the table, B&B is superior to BB3. We observe, for example, that the decrease in CPU time over BB3 is around 40% for the largest three instances. Also, our algorithm builds smaller search trees than BB3.

In our second experiment, we opt for three empirical dissimilarity matrices mentioned in Section 6.1. Table 2 summarizes the results of B&B and BB3 on the UDSP instances defined by these matrices. Its structure is the same as that of Table 1. In the instance names, the letter "M" stands for the Morse code matrix, "J" stands for the journal citations matrix, and "F" stands for the food item data. As before, the number following the letter indicates the dimension of the problem matrix. Comparing results in Tables 1 and 2, we find that for empirical data the superiority of our algorithm over BB3 is more pronounced. Basically, B&B performs significantly better than BB3 for each of the three data sets. For example, for the food-item dissimilarity matrix of size 30×30, B&B builds almost 10 times smaller search tree and is about 5 times faster than BB3.

Table 3 presents the results of the performance of B&B on the full Morse code dissimilarity matrix as well as on larger submatrices of the food-item dissimilarity matrix. Running BB3 on these UDSP instances is too expensive, so this algorithm is not included in the table. By analyzing the results in Tables 2 and 3, we find that, for $i \in \{22, \ldots, 35\}$, the CPU time taken by B&B to solve the problem with the $i \times i$ food-item submatrix is between 196 (for $i = 30$) and 282 (for $i = 27$) percent of that needed to solve the problem with the $(i-1) \times (i-1)$ food-item submatrix. For the submatrix of size 35×35, the CPU time taken by the algorithm was more than

Table 1: Performance on random problem instances.

Inst.	Objective function value	B&B		BB3	
		Number of nodes	Time	Number of nodes	Time
p20	9053452 (179755.4; 0.284)	1264871	4.4	1526725	5.6
p21	9887394 (187927.7; 0.285)	3304678	10.6	4169099	14.5
p22	12548324 (224540.6; 0.282)	6210311	20.5	8017112	29.0
p23	14632006 (239619.8; 0.274)	12979353	45.0	16624663	1:06.7
p24	16211210 (281585.9; 0.294)	31820463	1:53.3	42435561	2:55.5
p25	15877026 (312670.0; 0.330)	103208158	6:44.6	137525505	9:52.4
p26	20837730 (347147.8; 0.302)	200528002	13:35.4	267603807	20:43.6
p27	21464870 (358595.8; 0.311)	287794935	20:37.7	392405143	32:36.5
p28	23938264 (396136.0; 0.317)	581602877	44:09.3	812649051	1:13:05.4
p29	28613820 (453778.6; 0.315)	2047228373	2:50:57.2	3044704554	4:38:57.0
p30	30359134 (464556.9; 0.315)	3555327537	5:09:09.9	5216804247	8:55:04.6

Table 2: Performance on empirical dissimilarity matrices.

Inst.	Objective function value	B&B		BB3	
		Number of nodes	Time	Number of nodes	Time
M26	180577772 (1944889.1; 0.219)	108400003	7:43.2	201264785	16:14.3
J28	60653172 (716703.3; 0.249)	2019598593	3:48:00.6	6780517853	10:39:06.6
F20	19344116 (105143.2; 0.098)	236990	2.7	1971840	7.0
F21	22694178 (122997.9; 0.102)	568998	5.2	4716777	16.2
F22	26496098 (149594.9; 0.110)	1178415	10.6	9993241	35.7
F23	30853716 (175768.1; 0.116)	2524703	25.2	24357098	1:37.8
F24	35075372 (217471.2; 0.130)	6727692	1:00.1	51593000	3:30.4
F25	40091544 (249100.2; 0.134)	14560522	2:28.3	139580891	10:01.4
F26	45507780 (288604.8; 0.142)	40465339	6:55.3	298597411	23:18.8
F27	51859210 (333682.5; 0.148)	116161004	19:29.8	670492972	55:57.8
F28	58712130 (378864.2; 0.153)	253278124	44:35.8	1366152797	2:03:26.5
F29	66198704 (420398.1; 0.156)	530219761	1:42:26.5	3616257778	5:32:08.4
F30	74310052 (462935.3; 0.157)	972413816	3:20:56.6	9360343618	15:56:41.2

14 days. In fact, a solution of value 120526370 was found by the ITS heuristic in less than 1 s. The rest of the time was spent on proving its optimality. We also see from Table 3 that the UDSP with the full Morse code dissimilarity matrix required nearly 21 days to solve to proven optimality using our approach.

In Table 4, we display optimal solutions for the two largest UDSP instances we have tried. The second and fourth columns of the table contain the optimal permutations for these instances. The coordinates for objects, calculated using (4), are given in the third and fifth columns. The number in parenthesis in the fourth column shows the order in which the food items appear within Table 5.1 in the book by Hubert et al. [30]. From Table 4, we observe that only a half of the signals representing digits in the Morse code are placed at the end of the scale. Other such signals are positioned between signals encoding letters. We also see that shorter signals tend to occupy positions at the beginning of the permutation p.

In Table 5, we evaluate the effectiveness of tests used to fathom partial solutions. Computational results are reported for a selection of UDSP instances. The second, third, and fourth columns of the table present $100N_{\text{sym}}/N$, $100N_{\text{dom}}/N$, and $100N_{\text{UB}}/N$, where N_{sym} (respectively, N_{dom} and N_{UB}) is the number of partial permutations that are fathomed due to symmetry test (respectively, due to dominance test and due to upper bound test) in Step (3) of SelectObject, and $N = N_{\text{sym}} + N_{\text{dom}} + N_{\text{UB}}$. The last three columns give the same set of statistics for the BB3 algorithm. Inspection of Table 5 reveals that N_{dom} is much larger than both N_{sym} and N_{UB}. It can also be seen that N_{dom} values are consistently higher with BB3 than when using our approach. Comparing the other two tests, we observe that N_{sym} is larger than N_{UB} in all cases except when applying B&B to the food-item dissimilarity submatrices of size 20×20 and 25×25. Generally, we notice that the upper bound test is most successful for the UDSP instances defined by the food-item matrix.

We end this section with a discussion of two issues regarding the use of heuristic algorithms for the UDSP. From a practical point of view, heuristics, of course, are preferable to exact methods as they are fast and can be applied to large problem instances. If carefully designed, they are able to produce solutions of very high quality. The development of exact methods, on the other hand, is a theoretical avenue for coping with different problems. In the field of optimization, exact algorithms, for a problem of interest, provide not only a solution but also a certificate of its optimality. Therefore, one of possible applications of exact methods is to use them in

Table 3: Performance of B&B on larger empirical dissimilarity matrices.

Instance	Objective function value	Number of nodes	Time
F31	82221440 (519445.5; 0.164)	2469803824	8:57:03.8
F32	91389180 (570324.1; 0.166)	5040451368	19:44:12.7
F33	100367800 (641239.5; 0.174)	14318454604	55:24:20.0
F34	110065196 (710405.4; 0.180)	33035826208	129:36:50.0
F35	120526370 (784156.4; 0.185)	82986984742	342:56:45.6
M36	489111494 (4068295.6; 0.230)	243227368848	494:51:17.5

Table 4: Optimal solutions for the full Morse code dissimilarity matrix and the 35 × 35 food-item dissimilarity matrix.

i	Morse code full		Food item data	
	$p(i)$	$x_{p(i)}$	$p(i)$	$x_{p(i)}$
1	(E) •	0.00000	orange (3)	0.00000
2	(T) –	5.83333	watermelon (2)	0.28571
3	(I) ••	24.77778	apple (1)	0.28571
4	(A) •–	34.08333	banana (4)	1.28571
5	(N) –•	44.11111	pineapple (5)	1.62857
6	(M) – –	52.33333	lettuce (6)	21.25714
7	(S) • • •	70.30556	broccoli (7)	22.31429
8	(U) • • –	83.13889	carrots (8)	22.54286
9	(R) • – •	93.47222	corn (9)	22.88571
10	(W) • – –	102.97222	onions (10)	25.11429
11	(H) • • ••	114.44444	potato (11)	29.97143
12	(D) – • •	124.30556	rice (12)	51.62857
13	(K) – • –	131.52778	spaghetti (19)	58.97143
14	(V) • • •–	145.00000	bread (13)	64.48571
15	(5) • • • • •	152.44444	bagel (14)	69.85714
16	(4) • • • • –	159.91667	cereal (16)	72.11429
17	(F) • • –•	170.75000	oatmeal (15)	72.51429
18	(L) • – ••	182.25000	pancake (18)	76.37143
19	(B) – • ••	189.52778	muffin (17)	77.80000
20	(X) – • •–	199.55556	crackers (20)	88.97143
21	(6) – • • • •	205.66667	granola bar (21)	93.00000
22	(3) • • • – –	220.13889	pretzels (22)	100.74286
23	(C) – • –•	229.47222	nuts (24)	104.37143
24	(Y) – • ––	238.41667	popcorn (23)	107.62857
25	(7) – – • • •	249.30556	potato chips (25)	112.48571
26	(Z) – – ••	254.88889	doughnuts (26)	120.08571
27	(Q) – – •–	264.38889	pizza (31)	126.51429
28	(P) • – –•	270.83333	cookies (27)	136.88571
29	(J) • – ––	282.94444	chocolate bar (29)	139.40000
30	(G) – – •	292.47222	cake (28)	141.05714
31	(O) – – –	300.22222	pie (30)	143.54286
32	(2) • • – – –	310.50000	ice cream (32)	152.00000
33	(8) – – – • •	320.36111	yogurt (33)	157.11429
34	(1) • – – – –	333.50000	butter (34)	161.80000
35	(9) – – – – •	341.86111	cheese (35)	165.08571
36	(0) – – – – –	350.27778		

Table 5: Relative performance of the tests.

Instance	B & B			BB3		
	Sym.	Dom.	UB	Sym.	Dom.	UB
p20	11.58[a]	88.24[b]	0.18[c]	11.11[a]	88.73[b]	0.16[c]
p25	8.86[a]	90.87[b]	0.27[c]	8.48[a]	91.43[b]	0.09[c]
p30	7.01[a]	92.84[b]	0.15[c]	6.29[a]	93.64[b]	0.07[c]
M26	11.40[a]	88.47[b]	0.13[c]	10.44[a]	89.52[b]	0.04[c]
J28	8.55[a]	89.23[b]	2.22[c]	5.68[a]	94.26[b]	0.06[c]
F20	8.01[a]	75.63[b]	16.36[c]	13.25[a]	85.42[b]	1.33[c]
F25	7.04[a]	81.27[b]	11.69[c]	10.31[a]	89.39[b]	0.30[c]
F30	8.95[a]	83.91[b]	7.14[c]	7.74[a]	92.16[b]	0.10[c]
F35	8.53[a]	87.23[b]	4.24[c]	—[a]	—[b]	—[c]
M36	8.82[a]	91.14[b]	0.04[c]	—[a]	—[b]	—[c]

[a] Percentage of partial solutions fathomed due to symmetry test.
[b] Percentage of partial solutions fathomed due to dominance test.
[c] Percentage of partial solutions fathomed due to bound test.

the process of assessment of heuristic techniques. Specifically, optimal values can serve as a reference point with which the results of heuristics can be compared. However, obtaining an optimality certificate in many cases, including the UDSP, is very time consuming. In order to see the time differences between finding an optimal solution for UDSP with the help of heuristics and proving its optimality (as it is done by the branch-and-bound method described in this paper), we run the two heuristic algorithms, iterated tabu search and simulated annealing (SA) of Brusco [19], on all the problem instances used in the main experiment. For two random instances, namely, p21 and p25, SA was trapped in a local maximum. Therefore, we have enhanced our implementation of SA by allowing it to run in a multi start mode. This variation of SA located an optimal solution for the above-mentioned instances in the third and, respectively, second restarts. Both tested heuristics have proven to be very fast. In particular, ITS took in total only 2.95 seconds to find the global optima for all 30 instances we experimented with. Performing the same task by SA required 5.44 seconds. Such times are in huge contrast to what is seen in Tables 1–3. Basically, the long computation times recorded in these tables are the price we have to pay for obtaining not only a solution but also a guarantee of its optimality. We did not perform additional computational experiments with ITS and SA because investigation of heuristics for the UDSP is outside the scope of this paper.

Another issue concerns the use of heuristics for providing initial solutions for the branch-and-bound technique. We have employed the iterated tabu search procedure for this purpose. Our choice was influenced by the fact that we could adopt in ITS the same formulas as those used in the dominance test, which is a key component of our approach. Obviously, there are other heuristics one can apply in place of ITS. As it follows from the discussion in the preceding paragraph, a good candidate for this is the simulated annealing algorithm of Brusco [19]. Suppose that B&B is run twice on the same instance from different initial solutions, both of which are optimal. Then it should be clear that in both cases the same number of tree nodes is explored. This essentially means that any fast heuristic capable of providing optimal solutions is perfect to be used in the initialization step of B&B. An alternative strategy is to use in this step randomly generated permutations. The effects of this simple strategy are mixed. For example, for problem instances F25 and F27, the computation time increased from 148.3 and, respectively, 1169.8 seconds (see Table 2) to 413.3 and, respectively, 2062.5 seconds. However, for example, for p25 and M26, a somewhat different picture was observed. In the case of random initial solution, B&B took 407.5 and 468.7 seconds for these instances, respectively. The computation times in Table 1 for p25 and Table 2 for M26 are only marginally shorter. One possible explanation of this lies in the fact that, as Table 5 indicates, the percentage of partial solutions fathomed due to bound test for both p25 and M26 is quite low. However, in general, it is advantageous to use a heuristic for obtaining an initial solution for the branch-and-bound algorithm, especially because this solution can be produced at a negligible cost with respect to the rest of the computation.

7. Conclusions

In this paper we have presented a branch-and-bound algorithm for the least-squares unidimensional scaling problem. The algorithm incorporates a LAP-based upper bound test as well as a dominance test which allows reducing the redundancy in the search process drastically. The results of computational experiments indicate that the algorithm can be used to obtain provably optimal solutions for randomly generated dissimilarity matrices of size up to 30×30 and for empirical dissimilarity matrices of size up to 35×35. In particular, for the first time, the UDSP instance with the 36×36 Morse code dissimilarity matrix has been solved to guaranteed optimality. Another important instance is defined by the 45×45 food-item dissimilarity matrix D_{food}. It is a great challenge to the optimization community to develop an exact algorithm that will solve this instance. Our branch-and-bound algorithm is limited to submatrices of D_{food} of size

around 35×35. We also remark that optimal solutions for all UDSP instances used in our experiments can be found by applying heuristic algorithms such as iterated tabu search and simulated annealing procedures. The total time taken by each of these procedures to reach the global optima is only a few seconds. An advantage of the branch-and-bound algorithm is its ability to certify the optimality of the solution obtained in a rigorous manner.

Before closing, there are two more points worth of mention. First, we proposed a simple, yet effective, mechanism for making a decision about when it is useful to apply a bound test to the current partial solution and when such a test most probably will fail to discard this solution. For that purpose, a sample of partial solutions is evaluated in the preparation step of B&B. We believe that similar mechanisms may be operative in branch-and-bound algorithms for other difficult combinatorial optimization problems. Second, we developed an efficient procedure for exploring the pairwise interchange neighborhood of a solution in the search space. This procedure can be useful on its own in the design of heuristic algorithms for unidimensional scaling, especially those based on the local search principle.

Appendix

Derivation of (20)

We will deduce an expression for δ_{rk}, $1 \leqslant k < r \leqslant n$, assuming that p is a solution to the whole problem, that is, $p \in \Pi$. Relocating the object $p(r)$ from position r to position k gives the permutation $p' = (p(1), \ldots, p(k-1), p(r), p(k), \ldots, p(n))$. By using the definition of F we can write

$$\begin{aligned}
&\delta_{rk} \\
&\quad = F\left(p'\right) - F\left(p\right) \\
&\quad = \left(\sum_{l=1}^{k-1} d_{p(r)p(l)} - \sum_{l=k}^{n} d_{p(r)p(l)}\right)^2 - \left(\sum_{l=1}^{r-1} d_{p(r)p(l)} - \sum_{l=r+1}^{n} d_{p(r)p(l)}\right)^2 \\
&\qquad + \sum_{l=k}^{r-1}\left(\sum_{j=1}^{l-1} d_{p(l)p(j)} - \sum_{j=l+1}^{n} d_{p(l)p(j)} + 2d_{p(r)p(l)}\right)^2 \\
&\qquad - \sum_{l=k}^{r-1}\left(\sum_{j=1}^{l-1} d_{p(l)p(j)} - \sum_{j=l+1}^{n} d_{p(l)p(j)}\right)^2.
\end{aligned} \tag{A.1}$$

Let us denote the first two terms in (A.1) by δ' and the remaining two terms by $\delta'' = \sum_{l=k}^{r-1} \delta_l''$. It is easy to see that δ' stems from moving the object $p(r)$ to position k and δ'' stems from shifting objects $p(l)$, $l = k, \ldots, r-1$, by one position to the right. To get a simpler expression for δ', we perform the following manipulations:

$$\begin{aligned}
\delta' = &\left(\sum_{l=1}^{k-1} d_{p(r)p(l)}\right)^2 - 2\sum_{l=1}^{k-1} d_{p(r)p(l)}\left(\sum_{l=k}^{r-1} d_{p(r)p(l)} + \sum_{l=r+1}^{n} d_{p(r)p(l)}\right) \\
&+ \left(\sum_{l=k}^{r-1} d_{p(r)p(l)} + \sum_{l=r+1}^{n} d_{p(r)p(l)}\right)^2 \\
&- \left(\sum_{l=1}^{k-1} d_{p(r)p(l)} + \sum_{l=k}^{r-1} d_{p(r)p(l)}\right)^2 \\
&+ 2\left(\sum_{l=1}^{k-1} d_{p(r)p(l)} + \sum_{l=k}^{r-1} d_{p(r)p(l)}\right)\sum_{l=r+1}^{n} d_{p(r)p(l)} \\
&- \left(\sum_{l=r+1}^{n} d_{p(r)p(l)}\right)^2.
\end{aligned} \tag{A.2}$$

Continuing, we get

$$\begin{aligned}
\delta' = &\left(\sum_{l=1}^{k-1} d_{p(r)p(l)}\right)^2 - 2\sum_{l=1}^{k-1} d_{p(r)p(l)}\sum_{l=k}^{r-1} d_{p(r)p(l)} \\
&- 2\sum_{l=1}^{k-1} d_{p(r)p(l)}\sum_{l=r+1}^{n} d_{p(r)p(l)} + \left(\sum_{l=k}^{r-1} d_{p(r)p(l)}\right)^2 \\
&+ 2\sum_{l=k}^{r-1} d_{p(r)p(l)}\sum_{l=r+1}^{n} d_{p(r)p(l)} + \left(\sum_{l=r+1}^{n} d_{p(r)p(l)}\right)^2 \\
&- \left(\sum_{l=1}^{k-1} d_{p(r)p(l)}\right)^2 - 2\sum_{l=1}^{k-1} d_{p(r)p(l)}\sum_{l=k}^{r-1} d_{p(r)p(l)} \\
&- \left(\sum_{l=k}^{r-1} d_{p(r)p(l)}\right)^2 + 2\sum_{l=1}^{k-1} d_{p(r)p(l)}\sum_{l=r+1}^{n} d_{p(r)p(l)} \\
&+ 2\sum_{l=k}^{r-1} d_{p(r)p(l)}\sum_{l=r+1}^{n} d_{p(r)p(l)} - \left(\sum_{l=r+1}^{n} d_{p(r)p(l)}\right)^2.
\end{aligned} \tag{A.3}$$

Finally,

$$\begin{aligned}
\delta' &= -4\sum_{l=1}^{k-1} d_{p(r)p(l)}\sum_{l=k}^{r-1} d_{p(r)p(l)} + 4\sum_{l=k}^{r-1} d_{p(r)p(l)}\sum_{l=r+1}^{n} d_{p(r)p(l)} \\
&= -4\left(f_{p(r)} - \sum_{l=k}^{r-1} d_{p(r)p(l)}\right)\sum_{l=k}^{r-1} d_{p(r)p(l)} \\
&\quad + 4\sum_{l=k}^{r-1} d_{p(r)p(l)}\left(S_{p(r)} - f_{p(r)}\right) \\
&= 4\sum_{l=k}^{r-1} d_{p(r)p(l)}\left(\left(\sum_{l=k}^{r-1} d_{p(r)p(l)}\right) - 2f_{p(r)} + S_{p(r)}\right).
\end{aligned} \tag{A.4}$$

Similarly, for δ_l'' we have

$$\begin{aligned}\delta_l'' &= \left(\sum_{j=1}^{l-1} d_{p(l)p(j)} - \sum_{j=l+1}^{n} d_{p(l)p(j)}\right)^2 \\ &\quad + 4d_{p(r)p(l)}\left(\sum_{j=1}^{l-1} d_{p(l)p(j)} - \sum_{j=l+1}^{n} d_{p(l)p(j)}\right) \\ &\quad + 4d_{p(r)p(l)}^2 - \left(\sum_{j=1}^{l-1} d_{p(l)p(j)} - \sum_{j=l+1}^{n} d_{p(l)p(j)}\right)^2 \\ &= 4d_{p(r)p(l)}\left(f_{p(l)} - S_{p(l)} + f_{p(l)}\right) + 4d_{p(r)p(l)}^2 \\ &= 4d_{p(r)p(l)}\left(d_{p(r)p(l)} + 2f_{p(l)} - S_{p(l)}\right).\end{aligned} \tag{A.5}$$

By summing up (A.4) and (A.5) for $l = k, \ldots, r-1$ we obtain (20).

References

[1] I. Borg and P. J. F. Groenen, *Modern Multidimensional Scaling: Theory and Appplications*, Springer, New York, NY, USA, 2nd edition, 2005.

[2] L. Hubert, P. Arabie, and J. Meulman, *The Structural Representation of Proximity Matrices with MATLAB*, vol. 19, SIAM, Philadelphia, Pa, USA, 2006.

[3] S. Meisel and D. Mattfeld, "Synergies of Operations Research and Data Mining," *European Journal of Operational Research*, vol. 206, no. 1, pp. 1–10, 2010.

[4] G. Caraux and S. Pinloche, "PermutMatrix: a graphical environment to arrange gene expression profiles in optimal linear order," *Bioinformatics*, vol. 21, no. 7, pp. 1280–1281, 2005.

[5] M. K. Ng, E. S. Fung, and H.-Y. Liao, "Linkage disequilibrium map by unidimensional nonnegative scaling," in *Proceedings of the 1st International Symposium on Optimization and Systems Biology (OSB '07)*, pp. 302–308, Beijing, China, 2007.

[6] P. I. Armstrong, N. A. Fouad, J. Rounds, and L. Hubert, "Quantifying and interpreting group differences in interest profiles," *Journal of Career Assessment*, vol. 18, no. 2, pp. 115–132, 2010.

[7] L. Hubert and D. Steinley, "Agreement among supreme court justices: categorical versus continuous representation," *SIAM News*, vol. 38, no. 7, 2005.

[8] A. Dahabiah, J. Puentes, and B. Solaiman, "Digestive casebase mining based on possibility theory and linear unidimensional scaling," in *Proceedings of the 8th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED '09)*, pp. 218–223, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wis, USA, 2009.

[9] A. Dahabiah, J. Puentes, and B. Solaiman, "Possibilistic pattern recognition in a digestive database for mining imperfect data," *WSEAS Transactions on Systems*, vol. 8, no. 2, pp. 229–240, 2009.

[10] Y. Ge, Y. Leung, and J. Ma, "Unidimensional scaling classifier and its application to remotely sensed data," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS '05)*, pp. 3841–3844, July 2005.

[11] J. H. Bihn, M. Verhaagh, M. Brändle, and R. Brandl, "Do secondary forests act as refuges for old growth forest animals? Recovery of ant diversity in the Atlantic forest of Brazil," *Biological Conservation*, vol. 141, no. 3, pp. 733–743, 2008.

[12] D. Defays, "A short note on a method of seriation," *British Journal of Mathematical and Statistical Psychology*, vol. 31, no. 1, pp. 49–53, 1978.

[13] M. J. Brusco and S. Stahl, "Optimal least-squares unidimensional scaling: improved branch-and-bound procedures and comparison to dynamic programming," *Psychometrika*, vol. 70, no. 2, pp. 253–270, 2005.

[14] L. J. Hubert and P. Arabie, "Unidimensional scaling and combinatorial optimization," in *Multidimensional Data Analysis*, J. de Leeuw, W. J. Heiser, J. Meulman, and F. Critchley, Eds., pp. 181–196, DSWO Press, Leiden, The Netherlands, 1986.

[15] L. J. Hubert, P. Arabie, and J. J. Meulman, "Linear unidimensional scaling in the L_2-norm: basic optimization methods using MATLAB," *Journal of Classification*, vol. 19, no. 2, pp. 303–328, 2002.

[16] M. J. Brusco and S. Stahl, *Branch-and-Bound Applications in Combinatorial Data Analysis*, Springer Science + Business Media, New York, NY, USA, 2005.

[17] V. Pliner, "Metric unidimensional scaling and global optimization," *Journal of Classification*, vol. 13, no. 1, pp. 3–18, 1996.

[18] A. Murillo, J. F. Vera, and W. J. Heiser, "A permutation-translation simulated annealing algorithm for L_1 and L_2 unidimensional scaling," *Journal of Classification*, vol. 22, no. 1, pp. 119–138, 2005.

[19] M. J. Brusco, "On the performance of simulated annealing for large-scale L_2 unidimensional scaling," *Journal of Classification*, vol. 23, no. 2, pp. 255–268, 2006.

[20] M. J. Brusco, H. F. Köhn, and S. Stahl, "Heuristic implementation of dynamic programming for matrix permutation problems in combinatorial data analysis," *Psychometrika*, vol. 73, no. 3, pp. 503–522, 2008.

[21] G. Palubeckis, "Iterated tabu search for the maximum diversity problem," *Applied Mathematics and Computation*, vol. 189, no. 1, pp. 371–383, 2007.

[22] G. Palubeckis, "A new bounding procedure and an improved exact algorithm for the Max-2-SAT problem," *Applied Mathematics and Computation*, vol. 215, no. 3, pp. 1106–1117, 2009.

[23] J. Kluabwang, D. Puangdownreong, and S. Sujitjorn, "Multipath adaptive tabu search for a vehicle control problem," *Journal of Applied Mathematics*, vol. 2012, Article ID 731623, 20 pages, 2012.

[24] N. Sarasiri, K. Suthamno, and S. Sujitjorn, "Bacterial foraging-tabu search metaheuristics for identification of nonlinear friction model," *Journal of Applied Mathematics*, vol. 2012, Article ID 238563, 23 pages, 2012.

[25] A. Jouglet and J. Carlier, "Dominance rules in combinatorial optimization problems," *European Journal of Operational Research*, vol. 212, no. 3, pp. 433–444, 2011.

[26] E. Z. Rothkopf, "A measure of stimulus similarity and errors in some paired-associate learning tasks," *Journal of Experimental Psychology*, vol. 53, no. 2, pp. 94–101, 1957.

[27] I. Düntsch and G. Gediga, "Modal-Style Operators in Qualitative Data Analysis," Tech. Rep. CS-02-15, Department of Computer Science, Brock University, St. Catharines, Ontario, Canada, 2002.

[28] R. N. Shepard, "Analysis of proximities as a technique for the study of information processing in man," *Human factors*, vol. 5, pp. 33–48, 1963.

[29] P. J. F. Groenen and W. J. Heiser, "The tunneling method for global optimization in multidimensional scaling," *Psychometrika*, vol. 61, no. 3, pp. 529–550, 1996.

[30] L. Hubert, P. Arabie, and J. Meulman, *Combinatorial Data Analysis: Optimization by Dynamic Programming*, SIAM, Philadelphia, Pa, USA, 2001.

[31] B. H. Ross and G. L. Murphy, "Food for thought: cross-classification and category organization in a complex real-world domain," *Cognitive Psychology*, vol. 38, no. 4, pp. 495–553, 1999.