
Scatter Search for the Linear Ordering Problem

Vicente Campos^a, Manuel Laguna^b, and Rafael Martí^a

^a Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot, Valencia, Spain. Vicente.Campos@uv.es and Rafael.Marti@uv.es

^b Graduate School of Business and Administration, Campus Box 419, University of Colorado, Boulder, CO 80309-0419, USA. Manuel.Laguna@Colorado.edu

Latest Revision: February 2, 1999

Abstract — Scatter search is a population-based method that has recently been shown to yield promising outcomes for solving combinatorial and nonlinear optimization problems. Based on formulations originally proposed in the 1960s for combining decision rules and problem constraints, such as the surrogate constraint method, scatter search uses strategies for combining solution vectors that have proved effective in a variety of problem settings. In this paper, we present a scatter search implementation designed to find high quality solutions for the linear ordering problem. This NP-hard problem has a significant number of applications in practice. The LOP, for example, is equivalent to the so-called triangulation problem for input-output tables in economics. Our implementation goes beyond a simple exercise on applying scatter search, since it incorporates innovative mechanisms to combine solutions and to create a balance between quality and diversification in the reference set. Extensive computational experiments with input-output tables are used to assess the merit of our procedure.

1. Introduction

The goal of the research work presented in this paper is to expand the scatter search methodology by implementing innovative strategies within a scatter search procedure for the linear ordering problem. Before we address the contributions of our research in terms of advancing the scatter search methodology, we briefly discuss our problem setting.

The linear ordering problem (or LOP) has enjoyed a fair amount of research interest over the years, as documented in Grotschel, et al., 1984 and Chanas and Kobylanski, 1996. We use this problem as a test case for our strategies and search mechanisms. The problem can be described as follows.

Given a matrix of weights $E = \{e_{ij}\}_{m \times m}$, the LOP consists of finding a permutation p of the columns (and rows) in order to maximize the sum of the weights in the upper triangle. In mathematical terms, we seek to maximize:

$$C_E(p) = \sum_{i=1}^{m-1} \sum_{j=i+1}^m e_{p(i)p(j)}.$$

where $p(i)$ is the index of the column (and row) in position i in the permutation. Note that in the LOP, the permutation p provides the ordering of both the columns and the rows. The equivalent problem in graphs consists of finding, in a complete weighted graph, an acyclic tournament with a maximal sum of arc weights (Reinelt, 1985).

In economics, the LOP is equivalent to the so-called *triangulation problem for input-output tables*, which can be described as follows. The economy of a region (generally a country) is divided into m sectors and an $m \times m$ input-output table E is constructed where the entry e_{ij} denotes the amount of deliveries (in monetary value) from sector i to sector j in a given year. The triangulation problem then consists of simultaneously permuting the rows and columns of E , such that the sum of the entries above the main diagonal is as large as possible. An optimal solution then orders the sectors in such a way that the suppliers (i.e., sectors that tend to produce materials for other industries) come first followed by the consumers (i.e., sectors that tend to be final-product industries that deliver their output mostly to end users).

Instances of input-output tables from sectors in the European Economy can be found in the public-domain library LOLIB (1997). In the computational experiment section, we employ these problem instances to test the merit of our scatter search implementation.

2. Main Scatter Search Elements

Scatter search, from the standpoint of metaheuristic classification, may be viewed as an evolutionary (or also called population-based) algorithm that constructs solutions by combining others. It derives its foundations from strategies originally proposed for combining decision rules and constraints (in the context of integer programming). The goal of this methodology is to enable the implementation of solution procedures that can derive new solutions from combined elements in order to yield better solutions than those procedures that base their combinations only on a set of original elements.

Our solution method for the linear ordering problem is based on the scatter search template in Glover (1998). The procedure combines the following elements:

- a) Diversification Generator
- b) Improvement Method
- c) Reference Set Update Method
- d) Subset Generation Method
- e) Solution Combination Method

We now describe the implementation of each of these elements, as adapted in the context of the linear ordering problem.

a) Diversification Generator

The generator that we implemented is based on the notion of constructing solutions employing modified frequencies. The generator exploits the permutation nature of a linear ordering. A frequency counter is maintained to record the number of times an element i appears in position j . The frequency counters are used to penalize the “attractiveness” of an element with respect to a given position. To illustrate, suppose that the generator has created 30 solutions to be included in a population of solutions P . If 20 of the 30 solutions have element 3 in position 5, then the frequency counter $freq(3,5) = 20$. This frequency value is used to bias the potential assignment of element 3 in position 5 during subsequent constructions, and therefore, to induce diversification with respect to the solutions already in P .

The attractiveness of assigning element i to position j is given by the greedy function $G(i,j)$, as proposed in Becker (1967). We modify the value of $G(i,j)$ to reflect previous assignments of element i to position j , as follows:

$$G(i,j) = G(i,j) - \beta * (MaxG / MaxF) * Freq(i,j).$$

Where $MaxF$ is the maximum $Freq(i,j)$ value for all i and j , and $MaxG$ is the maximum $G(i,j)$ value for all i and j .

It is important to point out that in each iteration of the construction procedure, $G(i,j)$ is an adaptive function since its value depends on attributes of the unassigned elements. A pseudo-code of the diversification generator to produce a population P with $PSize$ solutions, appears in Figure 1.

Figure 1. Pseudo-code of diversification generator.

```

1.   $P = \emptyset$ .
2.   $Freq(i,j) = 0$ , for all  $i$  and  $j$ .
While ( $|P| < PSize$ ) do
3.   $p(i) = 0$  for all  $i$ .
4.   $N = \{ 1, \dots, m \}$  (The set of unassigned elements.)
5.   $j = MaxF = MaxG = 1$ .
   While ( $N \neq \emptyset$ ) do
6.   $G(i,j) = G(i,j) - \beta * (MaxG / MaxF) * Freq(i,j)$  for all  $i \in N$ .
7.   $i^* = \arg \max_i (G(i,j), i \in N)$ 
8.   $MaxG = G(i^*,j)$ 
9.   $N = N - \{ i^* \}$ 
10.  $p(i^*) = j$ 
11.  $j = j + 1$ 
   End while
12.  $Freq(i,p(i)) = Freq(i,p(i)) + 1$ , for all  $i$ .
13.  $MaxF = \max_{i,j} (Freq(i,j), i \in N, j \in \{1, \dots, m\})$ 
   If ( $p \notin P$ ) then  $P = P \cup \{p\}$ 
End while

```

The pseudo-code in Figure 1 is written using general mathematical notation. However, the actual implementation takes advantage of quick updating mechanisms that are hard to represent mathematically. For example, the value of $MaxF$ in step 13 can be maintained by keeping track of the updates of the frequency matrix $Freq$.

The performance of the diversification generator depends on the value of the parameter β . In order to determine effective values for this parameter, we created a measure of diversity for a set of solutions. The diversity measure is calculated as follows:

1. Calculate the median position of each sector i in the solutions in P .
2. Calculate the dissimilarity of each solution in the population with respect to the median position. The dissimilarity is calculated as the sum of the absolute difference between the position in the solution under consideration and the median solution.
3. Calculate d as the sum of all the individual dissimilarities.

To illustrate, suppose that P consists of the following three orderings: (A,B,C,D), (B,D,C,A), (C,B,A,D). Then the median position of sector A is 3, since it occupies positions 1, 3 and 4 in the given orderings. In the same way, the median positions of B, C and D are 2, 3 and 4. Note that the median positions might not induce an ordering, as in the case of this example. The dissimilarity of the first solution is then calculated as follows:

$$d_1 = |1-3| + |2-2| + |3-3| + |4-4| = 2.$$

In the same way, the dissimilarities of the other two solutions are $d_2 = 4$ and $d_3 = 2$. The diversity measure for P is then given by $d = 2+4+2 = 8$.

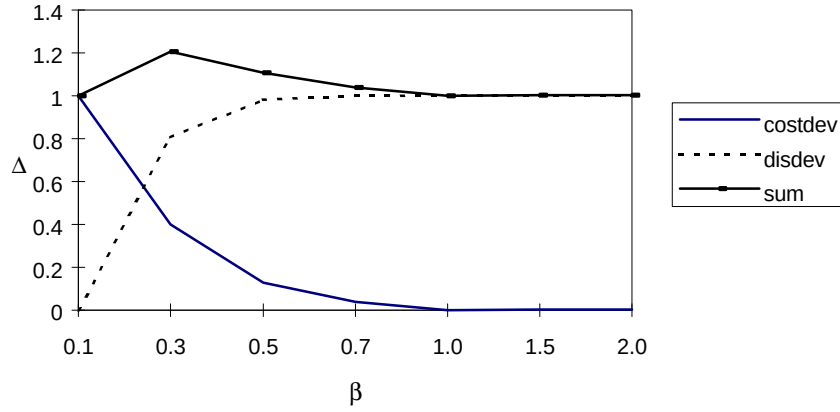
Since we are interested in finding a β -value that generates balanced populations, we use two relative measures: one based on the diversity value d , and one based on the objective function value C . Given a set of populations $\Pi = \{P_1, P_2, \dots, P_n\}$, we calculate $\Delta d(i)$ and $\Delta C(i)$ for each population P_i as follows:

$$\Delta d(i) = \frac{d(i) - d_{\min}}{d_{\max} - d_{\min}}, \text{ where } d(i) = \sum_{j \in P_i} d_j, \quad d_{\min} = \min_i \{d(i)\} \text{ and } d_{\max} = \max_i \{d(i)\};$$

$$\Delta C(i) = \frac{C(i) - C_{\min}}{C_{\max} - C_{\min}}, \text{ where } C(i) = \sum_{j \in P_i} C_j, \quad C_{\min} = \min_i \{C(i)\} \text{ and } C_{\max} = \max_i \{C(i)\}.$$

Figure 2 shows the values of our relative measures of quality and diversity when the diversification generator is executed with different β -values.

The legends in Figure 2 are such that $costdev = \Delta C$, $disdev = \Delta d$ and $sum = costdev + disdev$. This figure discloses that a value of 0.3 for β provides a good balance between diversity and quality in the population. As β increases, the diversity increases but the quality decreases. Therefore the value of sum is highest at $\beta = 0.3$, with a similar contribution from both relative deviation measures.

Figure 2. Diversity vs. quality for different β -values.

b) Improvement Method

The improvement method is based on the neighborhood search developed for the LOP in Laguna, Martí and Campos (1998). Two neighborhoods were considered in that study:

$$N_1 = \{p' : \text{INSERT_MOVE}(p(j), i), \text{ for } j = 1, \dots, m-1 \text{ and } i = j+1\}$$

$$N_2 = \{p' : \text{INSERT_MOVE}(p(j), i), \text{ for } j = 1, \dots, m \text{ and } i = 1, 2, \dots, j-1, j+1, \dots, m\}$$

N_1 consists of permutations that are reached by switching the positions of contiguous sectors $p(j)$ and $p(j+1)$. N_2 consists of all permutations resulting from executing general insertion moves, as defined above. In conjunction with these neighborhoods, two strategies are defined. The *best* strategy selects the move with the largest move value among all the moves in the neighborhood. The *first* strategy, on the other hand, scans the list of sectors (in the order given by the current permutation) in search for the first sector $p(f)$ whose movement results in an strictly positive move value (i.e., a move such that $C_E(p') > C_E(p)$). The move selected by the *first* strategy is then $\text{INSERT_MOVE}(p(f), i^*)$, where i^* is the position that maximizes $C_E(p')$. Note that for N_1 , $i^* = f+1$, while for N_2 , i^* is chosen from $i = 1, 2, \dots, f-1, f+1, \dots, m$. Therefore, the *first* strategy used in combination with N_1 is equivalent to searching for the first improving move in the neighborhood.

Combining the selection strategies with the neighborhood definitions results in four greedy local search procedures: *first*(N_1), *best*(N_1), *first*(N_2), and *best*(N_2). The experimentation in Laguna, Martí and Campos (1998) showed that the greedy procedure *first*(N_2) was the most effective. Based on this finding, we partition N_2 into m N_2^j neighborhoods

$$N_2^j = \{p' : \text{INSERT_MOVE}(p(j), i), i = 1, 2, \dots, j-1, j+1, \dots, m\}$$

associated with each sector $p(j)$, for $j = 1, \dots, m$. We therefore base our improvement method on choosing the best insertion associated with a given sector as proposed in Laguna, Martí and Campos (1998).

c) Reference Set Update Method

The Reference Set Update method accompanies each application of the Improvement Method. The Reference Update Method is generally examined right after the Improvement Method, because of its linking role with the Subset Generation Method. The update operation consists of maintaining a record of the b best solutions found, where the value of b is treated as a constant search parameter. The issues associated with this updating function are conceptually straightforward and are addressed with extensive detail in Glover (1998), where a concrete design for such a procedure is given.

Accompanying this design is a detailed pseudo-code description of the procedure for maintaining *RefSet* (i.e., the Reference Set), which is called the *RefSet* Update Routine. This routine is organized to handle vectors of 0-1 variables. We have adapted this *RefSet* Update Routine to handle permutation vectors, as needed in our current application.

d) Subset Generation Method

This Subset Generation procedure consists of creating different subsets X of *RefSet*, as a basis for implementing the subsequent combination method. The scatter search methodology prescribes that the set of combined solutions (i.e., the set of all combined solutions that the implementation intends to generate) is produced in its entirety at the point where X is created. Therefore, once a given subset X is created, there is no merit in creating it again. This creates a situation that differs noticeably from those considered in the context of genetic algorithms, where the combinations are typically determined by the spin of a roulette wheel.

The procedure seeks to generate subsets X of *RefSet* that have useful properties, while avoiding the duplication of subsets previously generated. The approach for doing this is organized to generate four different collections of subsets of *RefSet*, which Glover (1998) refers to as *SubSetType* = 1, 2, 3 and 4, with the following characteristics:

- SubSetType* = 1: all 2-element subsets.
- SubSetType* = 2: 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
- SubSetType* = 3: 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset.
- SubSetType* = 4: the subsets consisting of the best i elements, for $i = 5$ to b .

A central consideration of this design is that *RefSet* itself might not be static, because it might be changing as new solutions are added to replace old ones (when these new solutions qualify to be among the current b best solutions found). In our implementation, however, we maintain a static updating of *RefSet*, but use a broad definition of “best” for the membership in this set. In other words, we do not allow *RefSet* to dynamically change its size, but we use two criteria to allow solutions to become members of this set. One criterion is the quality of the solution (as given by the objective function value) and the other is the diversity of the solution (as given by the dissimilarity measure). In this sense, our definition of “best” is broader than one that considers only the value of the objective function.

e) Solution Combination Method

The Solution Combination Method as well as the Improvement Method are elements of scatter search that are context-dependent. Although it is possible to design “generic” combination procedures, it is more effective to base the design on specific characteristics of the problem setting. Our Solution Combination Method, which is applied to each subset generated in the previous step, uses a min-max construction based on votes.

The method scans (from left to right) each reference permutation in the subset, and uses the rule that each reference permutation votes for its first element that is still not included in the combined permutation (referred to as the “incipient element”). The voting determines the next element to enter the first still unassigned position of the combined permutation. This is a min-max rule in the sense that if any element of the reference permutation is chosen other than the incipient element, then it would increase the deviation between the reference and the combined permutations. Similarly, if the incipient element were placed later in the combined permutation than its next available position, this deviation would also increase. So the rule attempts to minimize the maximum deviation of the combined solution from the reference solution, subject to the fact that other reference solutions in the subset are also competing to contribute.

This voting scheme can be implemented using a couple of variations that depend on the way votes are modified:

- (V1) The vote of a given reference solution is weighted according to the incipient element’s position (referred to as the “incipient position”). A smaller incipient position gets a higher vote. For example, if the element in the first position of some reference permutation is not assigned to the combined permutation during the first 4 assignments, then the vote is weighted more heavily to increase the chances of having that element assigned to position 5 of the combined permutation. The rule emphasizes the preference to this assignment versus having a later occurring element of some other reference permutation (which is the incipient element for that other permutation) become assigned.
- (V2) A bias factor that gives more weight to the vote of a reference permutation with higher quality. Within the current organization of our scatter search implementation such a factor should be very slight because it is expected that high quality solutions will be strongly represented anyway.

3. Overall Procedure

The proposed procedure can be summarized as follows:

1. Use the diversification generator to create a set P of size $|P| = PSize$ of different solutions. That is, all the solutions in P are different from each other.
2. Apply the improvement method to P and retain all resulting solutions that are different. To guarantee that $|P| = PSize$, the diversification generator continues until $PSize$ different “improved” solutions are obtained.
3. Order the solutions in P according to their objective function value (where the best overall solution is first on the list).

For (*Iter* = 1 **to** *MaxIter*)

4. Build *RefSet* from *P*, with $|RefSet| = b_1 + b_2$ (i.e., $b = b_1 + b_2$). Take the first b_1 solutions in *P* and assign them to *RefSet*. For each solution x in $P - RefSet$, calculate the minimum dissimilarity $d(RefSet, x)$ to all solutions in *RefSet*. Select the solution x' with the maximum dissimilarity $d(RefSet, x')$ of all x in $P - RefSet$. Add x' to *RefSet*, until $|RefSet| = b_1 + b_2$. Make *SubsetCounter* = 0.

While (*SubsetCounter* < *MaxSubset*) **do**

5. Generate the next subset r from *RefSet* with the Subset Generation Method. This method generates one of four types of subsets with number of elements ranging from 2 to $|RefSet|$. Let subset $r = \{r_1, \dots, r_k\}$, for $2 \leq k \leq |RefSet|$.
6. Apply the Solution Combination Method to r to obtain a new solution s_r .
7. Apply the Improvement Method to s_r , to obtain the improved solution s_r^* .

If (s_r^* is not in *RefSet* and the objective function value of s_r^* is better than the objective function value of the worst element in *RefSet*) **then**

8. Add s_r^* to *RefSet* and delete the worst element currently in *RefSet*.
9. Make *SubsetCounter* = 0.

Else

10. Make *SubsetCounter* = *SubsetCounter* + 1.

End if

End while

If (*Iter* < *MaxIter*) **then**

11. Build a new set *P* using the diversification generator. Initialize the generation process with the best b_1 solutions currently in *RefSet*. That is, the first b_1 solutions in the new *P* are the best b_1 solutions in the current *RefSet*.

End if

End for

We now turn our attention to the computational experiments used to assess the merit of our scatter search implementations. Details of such experiments are provided in the next section.

4. Computational Experiments

The procedure described in the previous section was implemented in C, and all experiments were performed on a Pentium 166 Mhz personal computer. In the following set of experiments we compare the performance of our scatter search

implementation with three methods: Chanas and Kobylanski (1996), Tabu Search (Laguna, Martí and Campos, 1998) and a greedy procedure described below.

The greedy local search procedure is based on deleting a sector from its current position and inserting it in another position. An iteration of the algorithm consists of scanning the list of sectors in search for the first one whose movement results in an improvement of the objective function value. The algorithm ends when there is no improving move.

Preliminary tests were carried out in order to set the key parameters of the scatter search algorithm. After experimentation, the following values provide a balance between solution quality and run time: $PSize = 100$, $\beta = 0.3$, $b_1 = 10$ and $b_2 = 10$. Two different implementations are reported: the first one with $MaxIter = 1$ (SS), while the second one with $MaxIter = 10$ (SS10).

We have tested the procedures in two sets of instances: (1) the 49 instances in the public-domain library LOLIB (1997) which contains instances of input-output tables from sectors in the European Economy, and (2) 25 randomly generated instances. A uniform distribution with parameters (0, 25000) was used to generate the random instances of size 75.

To reduce the computational effort related to avoiding duplications in the population, we have implemented the following hash function to compare solutions and avoid duplications:

$$hash(p) = \sum_{i=1}^m ip(i)^2$$

We have empirically determined that two different solutions almost always have different hash values (except for some rare exceptions). Therefore, in our implementation, when two solutions have the same hash value we consider that both are the same and eliminate one of them from further consideration. This implementation results in average computational-time savings of about 7% in LOLIB instances and 1.5% in random instances when compared to a full duplication checking mechanism. The scatter search procedure yields significantly lower duplications in random instances than in real-world examples. Intuitively, this can be explained by the relationships between sectors in real-world examples, which tend to favor the generation of similar solutions.

Tables 1 and 2 show, for each procedure, the average objective function value, the average percent deviation from optimality, the number of optimal solutions, and the average CPU time in seconds. Since optimal solutions are not known for the random instances, the deviation in Table 2 is reported considering the best solution found during each experiment. Also for this table, the number of best solutions found is reported instead of the number of optimal solutions. We refer to Chanas and Kobylanski's method as CK, and as CK10 to the application of the method from 10 randomly generated initial solutions. The tabu search method will be denoted as TS.

Table 1. Results with LOLIB instances.

	greedy	CK	CK10	TS	SS	SS10
value	22033729.5	22018008.3	22040892.1	22040159.4	22041229.8	22041232.3
optima dev.	0.15%	0.15%	0.02%	0.04%	0.01%	0.01%
number of optima	11	11	27	30	42	43
run time (sec.)	0.01	0.10	1.06	0.33	3.82	14.28

Table 2. Results with random instances.

	greedy	CK	CK10	TS	SS	SS10
value	32713295.6	32678348.7	32800044.7	32890757.4	32903496.5	32908203.7
best dev.	0.60%	0.71%	0.34%	0.04%	0.02%	0.01%
number of best	0	0	0	12	10	16
run time (sec.)	0.01	0.65	6.56	1.38	7.45	38.68

The greedy procedure is clearly inferior in terms of solution quality, although given its simplicity, its performance is quite acceptable. The performance of the greedy and CK methods is very similar within each of the two problem sets, but their deviation from the optimal (or best) solutions is significantly higher in the case of random instances. TS and the SS variants, on the other hand, are quite robust, as evident by the negligible change in the deviation values from Table 1 to Table 2.

Tables 1 and 2 show that the best solution quality is obtained by the SS variants with an average percent deviation from optima of 0.01% in the LOLIB instances and 0.02% in the random instances. However, the SS variants employ more computational time than the other methods.

TS is very competitive, considering that is able to match 30 optima in the LOLIB and 12 best solutions in the random set, employing an average of 0.33 and 1.38 seconds, respectively. Also note the remarkable performance of SS10, which outperforms all other methods in terms of percent deviation as well as in terms of the number of optima matched.

5. Conclusions

The objective of our study has been to expand and advance the knowledge associated with the implementation of scatter search procedures. Unlike its close cousin, tabu search, scatter search has not yet become widely implemented and tested as a metaheuristic for solving complex optimization problems. Our study shows, however, that a careful scatter search implementation can be quite effective and rival the best procedures in the literature.

Acknowledgments

The authors wish to thank Fred Glover for his valuable comments on both content and presentation.

References

- Becker, O. (1967) "Das Helmstädtische Reihenfolgeproblem — die Effizienz verschiedener Näherungsverfahren" in *Computer uses in the Social Sciences*, Berichtener Working Conference, Wien, January 1967.
- Chanas, S. and P. Kobyłanski (1996) "A New Heuristic Algorithm Solving the Linear Ordering Problem," *Computational Optimization and Applications*, vol. 6, pp. 191-205.
- Crowston, W. B., F. Glover, G. L. Thompson and J. D. Trawick (1963) "Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules," ONR Research Memorandum No. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA.

Fisher, H. and G. L. Thompson (1963) "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules," *Industrial Scheduling*, J. F. Muth and G. L. Thompson (eds.) Prentice-Hall, pp. 225-251.

Glover, F. (1965) "A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, vol. 13, no. 6, pp. 879-919.

Glover, F. (1968) "Surrogate Constraints," *Operations Research*, vol. 16, pp. 741-749.

Glover, F. (1998) "A Template for Scatter Search and Path Relinking," in *Artificial Evolution, Lecture Notes in Computer Science* 1363, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (Eds.), Springer-Verlag, pp. 13-54.

Grotschel, M., M. Junger and G. Reinelt (1984), "A Cutting Plane Algorithm for the Linear Ordering Problem," *Operations Research*, vol. 32, no. 6, pp. 1195-1220.

Laguna, M., R. Martí and V. Campos (1998) "Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem," to appear in *Computers and Operations Research*.

LOLIB (1997) <http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/LOLIB/LOLIB.html>.

Reinelt, G. (1985) *The Linear Ordering Problem: Algorithms and Applications*, Research and Exposition in Mathematics, Vol. 8, H. H. Hofmann and R. Wille (Eds.), Heldermann Verlag Berlin.