J Comb Optim (2011) 22:572–593
DOI 10.1007/s10878-010-9306-x

Revised GRASP with path-relinking for the linear ordering problem

W. Art Chaovalitwongse · Carlos A.S. Oliveira · Bruno Chiarini · Panos M. Pardalos · Mauricio G.C. Resende

Published online: 2 March 2010

Abstract The linear ordering problem (LOP) is an $\mathcal{NP}$-hard combinatorial optimization problem with a wide range of applications in economics, archaeology, the social sciences, scheduling, and biology. It has, however, drawn little attention compared to other closely related problems such as the quadratic assignment problem and the traveling salesman problem. Due to its computational complexity, it is essential in practice to develop solution approaches to rapidly search for solution of high-quality. In this paper we propose a new algorithm based on a greedy randomized adaptive search procedure (GRASP) to efficiently solve the LOP. The algorithm is integrated with a Path-Relinking (PR) procedure and a new local search scheme. We tested our implementation on the set of 49 real-world instances of input-output tables (LOLIB instances) proposed in Reinelt (Linear ordering library (LOLIB) 2002).

W.A. Chaovalitwongse (✉)
Department of Industrial and Systems Engineering, Rutgers University, 96 Frelinghuysen Rd., Piscataway, NJ 08854, USA
e-mail: wchaoval@rci.rutgers.edu

C.A.S. Oliveira
Princeton Consultants Inc., 2 Research Way, Princeton, NJ 08540, USA
e-mail: oliveira@ufl.edu

B. Chiarini
P.O. Box 522274 Miami, FL 33152, USA
e-mail: bruno.chiarini@gmail.com

P.M. Pardalos
Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32601, USA
e-mail: pardalos@ufl.edu

M.G.C. Resende
AT&T Labs Research, 180 Park Avenue, Florham Park, NJ 07932, USA
e-mail: mgcr@research.att.com

In addition, we tested a set of 30 large randomly-generated instances proposed in Mitchell (Computational experience with an interior point cutting plane algorithm, Tech. rep., Mathematical Sciences, Rensellaer Polytechnic Institute, Troy, NY 12180-3590, USA 1997). Most of the LOLIB instances were solved to optimality within 0.87 seconds on average. The average gap for the randomly-generated instances was 0.0173% with an average running time of 21.98 seconds. The results indicate the efficiency and high-quality of the proposed heuristic procedure.

Keywords Linear ordering problem · Heuristic · GRASP · Path-relinking

1 Introduction

In this paper, we propose a new heuristic approach for solving the $\mathcal{NP}$-hard linear ordering problem (LOP), which has a wide range of applications in practice, especially the triangulation of input-output tables used to study the relationship among the sectors of an economy. The LOP is also known as the maximum acyclic subgraph problem, whose objective is to find the maximum weight subgraph that contains no cycles in a given weighted directed graph. The solution to this problem yields a linear ordering of the vertices in which all edges in the acyclic subgraph are forward edges.

The LOP can be formally stated as follows. Consider a set N of n objects and a permutation $\pi : N \to N$. Each permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ corresponds one to one to a linear ordering of the objects. Let e_{ij}, $i, j = 1, 2, \ldots, n$, be the cost of having i before j in the ordering, and E be the n-square matrix of costs. Then the *linear ordering problem* is to find a permutation π that maximizes the total cost

$$Z(\pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} e_{\pi(i)\pi(j)}. \tag{1}$$

Note that (1) is the sum of the elements above the diagonal of a matrix A, whose entry a_{ij} is resulting from a permutation π of the rows and columns of the matrix E. In other words, we have $A = XEX$, where X is the permutation matrix associated with the permutation π (Reinelt 1985). To be more precise, the LOP is a problem of finding a simultaneous permutation of the rows and columns of a matrix E such that the sum of the elements above the diagonal is maximal.

The LOP can also be modeled as a graph problem. Let $G(N, A)$ be a complete directed graph with node set N and arc set $A = \{(i, j) : i, j \in N \wedge i \neq j\}$. Let e_{ij} be the weight of arc (i, j). A spanning acyclic tournament in G induces a unique linear ordering of the node set N (Jünger 1985). A tournament is defined as a directed graph in which each pair of nodes is connected by exactly one arc, which is clearly necessary since either a sector i is before j or j is before i. The complexity of the maximum LOP can be easily proven to be $\mathcal{NP}$-hard by transforming it to the equivalent minimum weighted feedback arc set problem on G, which is a very well-known $\mathcal{NP}$-hard problem (Garey and Johnson 1979).

The LOP has an interesting symmetry property. If a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ is an optimal solution to the maximization version, then the reverse permutation $\overline{\pi} = (\pi(n), \pi(n-1), \ldots, \pi(1))$ is an optimal solution to the minimization

version. In fact, the LOP accepts a trivial $\frac{1}{2}$-approximation algorithm (Jünger 1985). Let π be an arbitrary permutation and $\overline{\pi}$ its reverse. It is easy to see that $Z(\pi)+Z(\overline{\pi})$ is a constant. Choose $\hat{\pi}$ such that $Z(\hat{\pi}) = \max\{Z(\pi), Z(\overline{\pi})\}$, then we get

$$\frac{Z(\pi^*) - Z(\hat{\pi})}{Z(\pi^*)} \leq \frac{1}{2},$$

where π^* is an optimal permutation and $Z(\pi^*) > 0$. No other approximation algorithm exists (Jünger 1985). It follows that any permutation is optimal in the unweighted version of the LOP.

1.1 Problem formulations

Like most combinatorial optimization problems, the LOP has many alternative formulations. The LOP can be expressed as an integer programming problem as follows. Let $G(N, A)$ be the complete directed graph associated with the LOP as shown in the previous section. Define

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in A', \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $A' \subset A$ is the arc set of the spanning acyclic tournament on G. Then the problem (1) can be formulated as the following integer program:

$$\max \quad \sum_{(i,j)\in A} e_{ij} x_{ij} \tag{3}$$

$$\text{s.t.} \quad x_{ij} + x_{ji} = 1 \quad \forall i, j \in N, i < j, \tag{4}$$

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \quad \forall i, j, k \in N, i \neq j, i \neq k, j \neq k, \tag{5}$$

$$x_{ij} \in \{0, 1\}.$$

The constraints in (4) define the tournament polytope. It can be proven that the 3-dicycle inequalities in (5) are sufficient to prevent any cycles (Reinelt 1985). These two constraint sets define the linear ordering polytope. There are $2\binom{n}{2}$ variables and $\binom{n}{2} + 2\binom{n}{3}$ constraints in this formulation.

The tournament constraints in (4) motivate the use of a single variable to represent the two possible ways, in which every pair of nodes can be connected. Let us substitute $x_{ji} = 1 - x_{ij}$, for every $i, j \in N, j > i$, then an equivalent integer programming formulation is given by

$$\max \quad \sum_{\{(i,j)\in A : i<j\}} e'_{ij} x_{ij} \tag{6}$$

$$\text{s.t.} \quad x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \forall i, j, k \in N, i \neq j, i \neq k, j \neq k, \tag{7}$$

$$x_{ij} + x_{jk} - x_{ik} \geq 0 \quad \forall i, j, k \in N, i \neq j, i \neq k, j \neq k, \tag{8}$$

$$x_{ij} \in \{0, 1\},$$

where $e'_{ij} = e_{ij} - e_{ji}$. With the reformulation technique, this integer program has only $\binom{n}{2}$ variables and $2\binom{n}{3}$ constraints.

Finally, the linear ordering problem can be formulated as a quadratic assignment problem (QAP), which is well-known to be $\mathcal{NP}$-complete. The QAP can be stated as follows: *Given a positive integer n, and two $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$ with nonnegative entries, find a permutation $p = (p(1), \ldots, p(n))$ of the set $\{1, 2, \ldots, n\}$ that minimizes*

$$C(p) \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{p(i)p(j)}.$$

The QAP can be formulated in several equivalent forms. One of the most common formulations is the following quadratic zero-one programming problem:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} a_{ij} b_{lk} x_{ik} x_{jl} \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, \ldots, n), \\
& \sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, \ldots, n), \\
& x_{ij} \in \{0, 1\} \quad (i, j = 1, \ldots, n).
\end{aligned} \tag{9}$$

The LOP can be transformed to the QAP by using the matrix of weights E of the LOP as the distance matrix of the QAP. Then, the flow matrix $A = \{a_{ij}\}$ is constructed as follows, $a_{ij} = -1$ if $i < j$ and $a_{ij} = 0$, otherwise (Burkard et al. 1998).

1.2 Application in economics

The main application of the LOP is the triangulation of input-output models in economics. The input-output models are primarily used by economists and policy-makers to obtain a systematic description of interrelations among the sectors (Leontief 1986). Specifically, the input-output models provide the impact of changes of an economic variable through transactions among the sectors of an economy. An input-output model begins by dividing the economy of a country (or region) into a specified number of sectors. Then a table is constructed, where the entries are the total transactions between every pair of sectors. The total output (or input) of a sector can be obtained by summing the entries on the corresponding row (or column). The resulting table thus summarizes the interdependence among the economic sectors. Structural properties of the input-output tables may not be apparent. A particular choice in the order of the sectors used in constructing the table might conceal an otherwise evident structure. These features are revealed by a process called *triangulation*, whose objective is to find a *hierarchy* of the sectors such that those who are predominantly producers will appear first, while those who are mainly consumers will appear last.

The degree to which an economic structure "agrees" with a hierarchy of the sectors is called *linearity*. In a perfectly linear economy, the flow of goods "cascades" from the upper sectors to the lower sectors of the hierarchic ordering. If we arrange the rows and columns of the input-output matrix according to the hierarchy, the linearity would be reflected by a matrix that has an upper triangular structure; that is, all entries below the diagonal would be zero. On the other hand, if there is a flow of goods back to the upper sectors, it would be reflected by positive values on the entries below the diagonal. This leads to the definition of a quantitative measure of linearity. Let n denote the number of sectors and $E = \{e_{ij}\}$ be the n-square matrix representing the input-output table. Assume that the rows and columns have been arranged according to the hierarchy. Then, the linearity of an economy is given by

$$\lambda = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} e_{ij}}{\sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} e_{ij}}. \tag{10}$$

That is, the linearity is the ratio of the sum of the elements above the diagonal to the sum of all elements (except the diagonal). It follows that $\lambda = 1$ for a perfectly linear economy. Typical linearity values are 70% and 90% for a highly developed and underdeveloped economy, respectively (Leontief 1986). In addition to the knowledge of a hierarchy of sectors, the *triangulation of an input-output table* is the process of finding a hierarchy of sectors among all possible orderings. It is obvious that such ordering is the one that most closely resembles an upper triangular matrix, and thus has the maximum value of λ. Note that the denominator of (10) is constant. Additionally, every ordering is a permutation of the sectors and can be applied to both the rows and columns of the input-output matrix. Therefore, we can state the *triangulation problem* as that of finding a permutation of the rows and columns such that the sum of the elements above the diagonal is maximum. Mathematically, this is equivalent to an LOP.

1.3 Other applications

In this section, we discuss a few other applications of the LOP besides that in economics (see Reinelt 1985 for an extensive survey). The following problems are application areas that the LOP has been applied.

Personnel Ranking Applications: Consider the problem of having a group of people rank n objects. Each individual in the group is asked to express their preference with respect to every possible pair. If we let e_{ij} be the number of people who preferred i to j, the solution to the corresponding LOP is the ranking that most likely reflects the preferences of the group.

Sports Applications: A similar application can be found in the context of sports. For example, consider a tournament of n teams in which every team plays against every other team. Let e_{ij} be the score of the match between i and j if i wins, and 0 otherwise. The ranking obtained by the LOP is considered to be the one that most closely reflects the "true" performance of the teams. Still, it has not gained support for its implementation, probably because the outcome of a particular match is not closely related to the result in the ranking.

Archaeology Applications: The LOP is used to determine the "most probable" chronological ordering of a set of artifacts recovered from different sites. Samples belonging to various time periods are given a value based on their distance to the surface. The objective is to aggregate the data and determine an ordering of the artifacts.

Wireless Applications: In data broadcast, to reduce the access latency for processing the complex query, it is beneficial to place the data accessed in a query close to each other on the broadcast channel (Lee et al. 2002). The LOP is used to optimize the performance of query processing by determining the best allocation of the data on the broadcast channel such that frequently co-accessed data are not only allocated close to each other. The LOP enables the data broadcast business to be widely expanded in the near future, where clients are expected to perform complex queries or transactions on the broadcast data.

1.4 Paper organization

The remainder of this paper is organized as follows. In Sect. 2, we give an overview discussion of some previous work. In Sect. 3, we give an introduction describing the GRASP and path-relinking framework and later describe a detailed implementation of the Revised GRASP with Path-Relinking algorithm. The computational results are given in Sect. 4. Section 5 gives some concluding remarks and a discussion of practical issues.

2 Previous work

2.1 Exact methods

A common approach in the LOP literature is the use of cutting plane algorithms (Grötschel et al. 1984; Reinelt 1985). The goal is to obtain an approximate description of the convex hull of the solution set by introducing valid inequalities that are violated by current fractional solutions, which are added to the set of inequalities of the current linear programming problem. Jünger (1985) studied the structure of a LOP-related problem, the acyclic directed subgraph problem, which is equivalent to the LOP if we restrict the solution to span the entire node set. Reinelt (1985) introduced facets induced by subgraphs. Bolotashvili et al. (1999) extended Reinelt's results by introducing a generalized method to generate new facets. A complete characterization has only been obtained for very small problems, $n = 7$ (Christof and Reinelt 1997). In fact, we know that unless $\mathcal{P} = \mathcal{NP}$ there exist an exponential number of such facets. However, research in this area has resulted in valid inequalities that improve the performance of branch-and-cut algorithms (Reinelt 1985). There are several other studies in the facet defining inequalities for the LOP polytope (Grötschel et al. 1985; Reinelt 1993; Leung and Lee 1994; Bolotashvili et al. 1999).

Although cutting plane methods provide a way to obtain an optimal solution to a linear ordering problem, it is now possible to solve linear ordering problems of a size where these linear programming problems can be solved more efficiently by using an interior point method than by using simplex. The first authors to discuss an

interior point cutting plane algorithm for the linear ordering problem were Mitchell (1997), in which the interior point method is integrated with a simplex cutting plane method. The solution given by a interior point algorithm is used as a starting point for a simplex cutting plane algorithm (Mitchell and Borchers 2000). They solved all the considered problem instances to optimality.

2.2 Approximation algorithms

Although the LOP is $\mathcal{NP}$-hard, the solution can be estimated to the optimality within a factor of $1/2$. It is not known whether the maximum can be estimated to a better factor using a polynomial-time algorithm. Hansen (1989) presented $O(\log^2 n)$-approximation algorithms for the LOP (also called minimum linear arrangement problem) and for the more general problem of graph embeddings in d-dimensional meshes. Even et al. (2000) applied the recursive decomposition technique to obtain polynomial time $O(\log n \log\log n)$-approximation algorithms for the LOP. In a later study, it was shown that the best previous approximation bound for the LOP can be improved to a polynomial time $O(\log n)$-approximation algorithm (Rao and Richa 2004). However, it was recently shown that polyhedral relaxations of the LOP cannot be used to approximate the problem to within a factor better than 1/2 (Newman 2000). This result was achieved by demonstrating that the integrality gap of the polyhedral relaxations is 2 on random graphs with uniform edge probability $p = \frac{2^{\sqrt{\log n}}}{n}$, where n is the number of vertices. This result is followed by another study showing that approximating the LOP to within better than $\frac{65}{66}$ is $\mathcal{NP}$-hard (Newman and Vempala 2001). In a more recent study, a new semidefinite programming relaxation for the linear ordering problem is proposed by showing that if a random graph is chosen with uniform edge probability $p = \frac{d}{n}$, where $d = \omega(1)$, then the gap between the semidefinite relaxation and the integral optimal is at most 1.64, with a high probability (Newman 2004). A semidefinite program for the LOP derived in that study is similar to the semidefinite program used in the Goemans-Williamson algorithm to approximate the maximum cut problem (Goemans and Williamson 1995) but different objective functions are used.

2.3 Heuristic methods

Heuristic methods such as *GRASP, tabu search, simulated annealing, genetic search*, and *evolution strategies* have shown to able to efficiently find high quality solutions to many combinatorial and global optimization problems, by exploring the solutions space more thoroughly to find a good solution. A recent survey on multi-start heuristic algorithms for global optimization is given by Martí (2003). One of the earliest heuristics for the LOP was proposed in Chenery and Watanabe (1958) in the context of the triangulation of input-output tables. Given a sector i, the ratio of total input to the total output

$$u_i = \frac{\sum_{k=1}^{n} c_{ik}}{\sum_{k=1}^{n} c_{ki}} \tag{11}$$

is used to arrange the sectors in the order of decreasing u_i. The use of (11) gives a fairly good ordering considering its simplicity. Based on the symmetry property mentioned in Sect. 1.2, Chanas and Kobylański (1996) developed a heuristic that performs a sequence of optimal insertions and reversals. Laguna et al. (1999) developed an algorithm based on tabu search. They analyzed several intensification and diversification techniques, and compared their algorithm with that of Chanas and Kobylański (1996). Campos et al. (2001) used a scatter search approach. A correction term based on the frequency by which an object i appears in a particular position in the ordering is added to 11 to reflect previous solutions. The preliminary version of this paper can be found in Campos et al. (1998). Variable Neighborhood Search for the LOP is proposed in González and Pérez-Brito (2001). In a later study, a hybrid genetic algorithm, which includes the selection of crossover/mutation operators, accelerating the local search module and tuning the parameters, for the LOP is proposed in Huang and Lim (2003). A Lagrangian-based heuristic approach is proposed in Belloni and Lucena (2004). This approach is embedded within a Lagrangian Relaxation framework and started with a construction phase. Solutions are then subsequently improved by Subgradient Optimization procedure. Greistorfer (2004) investigated several versions of a tabu scatter search heuristic to solve permutation type optimization problems including the LOP. The main idea of this method is to incorporate the following procedures: the input and output procedures used maintain the solution pool and determine the transfer of elite solutions, and a solution combination method used to effectively combine a set of elite solutions. In the most recent study, Campos et al. (2005) developed a general-purpose heuristic based on the scatter-search and tabu-search methodologies for permutations problems. The proposed technique treats the objective-function evaluation as a black box, which makes the search algorithm context-independent.

3 A GRASP with path-relinking algorithm

3.1 Introduction to GRASP and path-relinking

Many problems in combinatorial optimization are inherently intractable. In other cases, we might be able to obtain a global optimal solution but with a running time that exceeds what it is demanded in practice. Therefore, researchers and practitioners usually try to develop algorithms that can generate "good" solutions within an acceptable time limit. Most hard problems in combinatorial optimization require the use of heuristics to obtain approximate solutions due to their inherent intractability. In this case, we are interested in finding solutions that are close "enough" to the optimal value at a low computational cost.

Heuristics, as opposed to approximation algorithms, do not give a guaranteed quality of the obtained solutions. Nevertheless, the flexibility we have in developing heuristics allows us to exploit the special structure of the problem, tailoring the existing methods, and resulting in very well performing algorithms. The quality of a heuristic, however, must be validated by extensive testing. *GRASP* (*Greedy Randomized Adaptive Search Procedure*), which is an iterative restart approach, has proven to

Fig. 1 A generic GRASP pseudo-code

```
Procedure GRASP(RCLSize, StoppingCondition)
1    BestSolutionFound = ∅;
2    while StoppingCondition not satisfied, do
3        x = ConstructGreedyRandomizedSolution(RCLSize);
4        LocalSearch(x);
5        UpdateSolution(BestSolutionFound, x);
6    end;
7    return BestSolutionFound;
end;
```

be one of the most effective heuristics to date. In this paper, we developed a GRASP-based algorithm for the LOP, offering a significant improvement on the computational time and quality of solution compared to previous heuristics. In the next section, we discuss the basic principles for implementing a new local search scheme and Path-Relinking in the GRASP framework. Since its inception by Feo and Resende in the late 1980s, GRASP has been successfully used in many applications. In 1995, the authors formally introduced GRASP as a framework for the development of new heuristics (Feo and Resende 1995). For a recent extensive annotated bibliography of GRASP applications, see Festa and Resende (2000).

Each iteration of GRASP consists of two phases: a *construction phase* in which we seek to obtain a feasible solution, and a *local search phase* that attempts to improve the solution. Figure 1 shows a generic pseudo-code of GRASP. In the construction phase, we iteratively build a solution by randomly selecting objects from a restricted candidate list (RCL). At each step, we form the RCL by selecting those objects with the highest measures of attractiveness, we select a random object from the RCL, and adapt the greedy function to reflect the addition to the solution. The size of the list is typically restricted in one of two ways: by quality, when we choose the elements based on a threshold on the greedy function, or by cardinality. In the literature, they are often referred to as α and β, respectively. The size of the RCL controls the degree of greediness and randomness of the construction phase. A null RCL results in a purely greedy solution whereas a RCL size equal to the size of the problem yields a purely random solution.

After a solution is constructed, we attempt to improve it by performing a local search in an appropriately defined neighborhood. Given a solution x, we explore the neighborhood $N(x)$ aspiring to find a local (global) optimal solution. Although larger neighborhoods increase the probability of finding the global optimum, local search algorithms are often computationally expensive and thus careful consideration must be given to the election of the neighborhood. It is in this part of GRASP where the particular properties of the problem in question can be exploited to develop schemes that can provide an intensification of the search, while not compromising its running time. Finally, the best solution is updated if necessary with the newly found solution. The procedure is repeated until a stopping condition is met—for example, number of iterations, running time, etc.

Path-relinking was introduced in Glover and Laguna (1997) as a method to integrate intensification and diversification to tabu search. It explores the trajectories linking two good-quality solutions, starting at an initial solution and moving towards a guiding solution. A set of good quality solutions referred to as the *elite list* is stored,

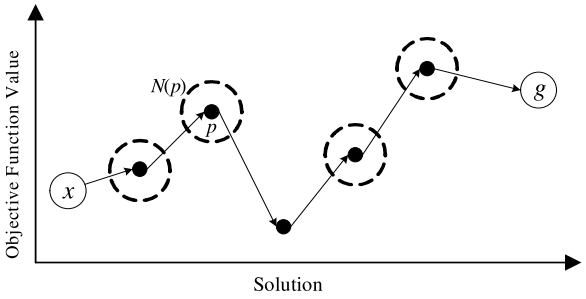

Fig. 2 A visualization of a path-relinking procedure. Given the initial solution s and the guiding solution g, we iteratively explore the trajectory linking these solutions, checking for any improvements on the way

from which the guiding solution is usually taken from. It generates new solutions by exploring routes that connect high-quality solutions by starting from one of these solutions, so-called *initiating solution,* and generating a path in the neighborhood space that leads toward the other solutions, a so-called *guiding solution.* This is completed by selecting moves that introduce attributes contained in the guiding solutions. Path relinking is a directly-focused instance of a strategy that seeks to fit in features of high quality solutions. On the other hand, instead of using an incentive that supports the inclusion of such attributes, the path relinking approach subordinates all other considerations to the goal of choosing moves that initiate the attributes of the guiding solutions, in order to generate a *good attribute composition* in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from the restricted set of moves that incorporate a maximum number of the attributes of guiding solutions (see Fig. 2).

3.2 Revised GRASP with path-relinking

Laguna and Martí (1998) were the first to combine GRASP with a path-relinking procedure, essentially adding memory to a procedure that would otherwise be a multi-start algorithm. In this paper, we propose a new heuristic that integrates GRASP with path-relinking for the linear ordering problem. In addition, we also revise the GRASP procedure based on the technique suggested in Chaovalitwongse et al. (2003). The GRASP procedure was revised by adding an ad hoc neighborhood greedy search in the construction phase. The revised procedure has been proven to make the best solution converged much faster (Chaovalitwongse et al. 2003). Figure 3 illustrates a flowchart of GRASP with path-relinking procedure. Figure 4 shows the pseudo-code for the generic implementation of GRASP with Path-Relinking.

For the LOP, the measure of attractiveness for each object consists of the difference between the row and column sums for the object, given by

$$d_i = \sum_{k=1}^{n} (e_{ik} - e_{ki}), \quad i = 1, 2, \ldots, n. \tag{12}$$

In the context of the LOP, (12) represents the net gain by substituting (permuting) two orders. In earlier experimentations with a GRASP algorithm (Chiarini et al. 2004), we compared the use of (12) with the ratios as defined in (11). Although we have not

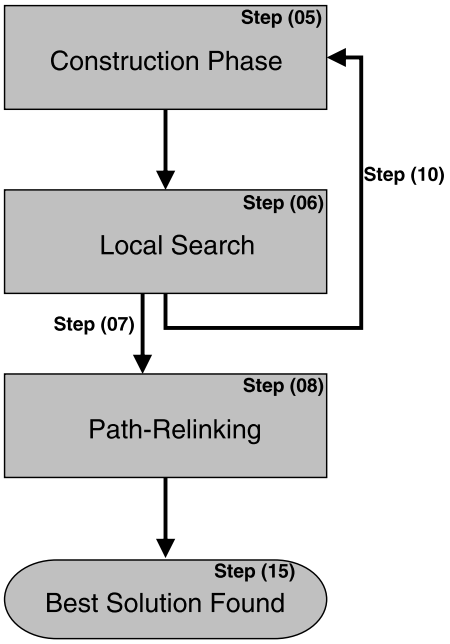

Fig. 3 A flowchart of GRASP with path-relinking procedure. The steps shown in this figure correspond to the ones in Fig. 4

Fig. 4 The revised GRASP with path-relinking pseudo-code

```
Procedure GRASP-PR(β, ρ, MaxIteration)
01    BestSolutionFound = ∅;
02    EliteList = ∅;
03    nNonImprovingIt = 0;
04    for k = 1, 2, ...,MaxIteration
05        x = ConstructGreedyRandomizedSolution(β);
06        LocalSearch(x);
07        if x is better than worse solution in EliteList
08            DoPathRelinking(EliteList, x);
09            nNonImprovingIt = 0;
10        else if nNonImprovingIt > γ
11            AdjustEliteList(EliteList, ρ);
12            DoPathRelinking(EliteList, x);
13            nNonImprovingIt = 0;
14        else nNonImprovingIt = nNonImprovingIt + 1;
13        UpdateSolution(BestSolutionFound, x);
14    rof;
15    return BestSolutionFound;
end;
```

observed significant differences in the quality of the solutions, adapting the greedy function when ratios are used is computationally more expensive. The use of (11) demands $\mathcal{O}(n^2)$ time to update the greedy function whereas (12) requires $\mathcal{O}(n)$ time. For this reason, based on our experience, we will use (12) as our greedy function.

As apposed to a generic implementation of GRASP with Path-Relinking, we herein propose a revised implementation. Since, LOPs usually have many alternative solutions (for both optimal and suboptimal) with the same objective function value, it

Fig. 5 The GRASP construction phase

```
procedure ConstructGreedyRandomizedSolution(β)
1    Solution = ∅, RCL = ∅;
2    while |Solution| < N
3        MakeRCL(RCL, β);
4        s = SelectElementAtRandom(RCL);
5        Insert(Solution, s);
6        NeighborGreedySearch(s);
7        AdaptGreedyFunction(s);
8    end;
end;
```

may occur at some point in the algorithm that the elite list becomes mostly populated by alternative solutions. Thus, it is increasingly difficult to enter a path-relinking as the best solution found approaches the optimal. To solve this problem, we attempt to avoid such situations by adapting the elite list and forcing a path-relinking procedure after a certain number of non-improving iterations. Next, we proceed to discuss in detail the different components of the algorithm.

3.2.1 Construction phase

In this phase, we will construct a set of good feasible solutions based on our greedy function. We start the procedure by creating the restricted candidate list (RCL) (see Fig. 5), which is the list of good candidates to be inserted to a solution. The parameter β of GRASP determines the cardinality limit on the RCL—i.e., the number of elements in RCL. The value of β defines the trade-off between the diversity and greediness of GRASP. Larger values of β will lead to greater diversity of the element candidates but if β is too large, then it will increase a chance of constructing many lower-quality solutions. The best value of β is usually determined by extensive testings. Once the RCL is created, an element is selected at random. After the selection, we proceed to insert the selected element in the partial solution. A conventional GRASP implementation would simply append the recently selected object s to the end of the partial solution. Instead, we added a procedure named "Insert" (line 5 on Fig. 5) that seeks to insert the object in an optimal position. More precisely, let $T_k = (t_1, t_2, \ldots, t_k)$, $k = 1, 2, \ldots, n$, denote the current (partial) solution obtained after k steps. The Insert operation intercalates the most recently selected object s in T_k in the position r that maximizes

$$\Delta(i, T_k) = \sum_{j=1}^{r-1} e_{t_j i} + \sum_{j=r}^{k} e_{i t_j} \tag{13}$$

breaking ties arbitrarily. First introduced in Chanas and Kobylański (1996) as part of their heuristic, it can be considered as a very efficient local search procedure in a relatively small neighborhood. In fact, it can be implemented in $\mathcal{O}(k)$. After the insertion, the algorithm then performs a small neighborhood greedy search as proposed in Chaovalitwongse et al. (2003). A step of the construction phase finalizes with the task of adapting the greedy function. The row and column corresponding to the object s are removed from the matrix, and the attractiveness (12) of the objects is updated.

Fig. 6 Local Search algorithm

```
procedure LocalSearch()
01    max ← −∞
02    for i from 1 to n
03        for j from 1 to n
04            delta ← −e'_{π(i)π(j)}
05            for k from 1 to n − 1
06                delta ← delta + (e'_{π(i)π(k)} + e'_{π(k)π(i)})
07            end
08            if delta > max
09                max ← delta; besti ← i; bestj ← j
12            end
13        end
14    end
15    return (max, besti, bestj)
end;
```

We set $d_s = -M$, where M is a large positive value, and re-sort the top $n - k$ objects that have yet to be selected. The procedure continues until a solution is constructed.

3.2.2 Local search

Local search is one of the critical parts of the GRASP algorithm. The local search method uses a greedy algorithm to explore the neighborhood of the current solution with the goal of improving it as much as possible.

After a set of good solutions is constructed, we apply a 2-exchange neighborhood (2-opt exchange) as a local search procedure. Given a solution π, its 2-exchange neighborhood $N(\pi)$ consists of all the solutions obtained by permuting the position of two objects in the ordering—i.e., if $\pi = (3, 1, 2)$, then $N(\pi) = \{(1, 3, 2), (2, 1, 3), (3, 2, 1)\}$. Clearly, for a problem of size n, $|N(\pi)| = \binom{n}{2}$.

The simplest way to implement local search for the LOP is to find, for each possible pair of positions i and j, the value of the objective function when positions i and j are exchanged in the permutation. However, directly calculating the objective function for each perturbation of the original solution is inefficient.

A better way to do local search is to calculate only the improvement achieved by exchanging a pair of positions (i, j). Consider a solution π and two objects $\pi(i)$ and $\pi(j)$ located in positions i and j respectively. For simplicity assume that $i < j$. The change in the objective function for an exchange of objects $\pi(i)$ and $\pi(j)$ is

$$\Delta Z(\pi, i, j) = -e^{\pi}_{i,j} - \sum_{k=i+1}^{j-1} (e^{\pi}_{i,k} + e^{\pi}_{k,j}) \tag{14}$$

where $e^{\pi}_{i,j} = e_{\pi(i)\pi(j)} - e_{\pi(j)\pi(i)}$. Using this formula we can derive a more efficient algorithm, displayed on Fig. 6.

At completion, the local search would have exchanged the pair of objects that maximizes (14). The operations of exploring the neighborhood and performing the exchange can be implemented in $\mathcal{O}(n^3)$.

3.2.3 Using dynamic programming to calculate improvements

The algorithm described above can compute an optimum exchange pair for local search, but with unnecessarily high complexity. While Algorithm 6 has time complexity $O(n^3)$, which is better than the $O(n^4)$ of the naive algorithm, this can be impractical for a GRASP implementation, which repeatedly uses local search to improve solutions.

To avoid this high computational complexity, we devise a dynamic programming strategy for calculating a local optimum for the LOP. The method is based on computing the improvement due to the exchange of positions (i, j) in terms of the improvements due to the exchange of neighboring positions. In this way we can leverage previous calculations and avoid duplication of effort.

First, we prove a basic result that shows the recursive nature of improvements in the local search algorithm. The result directly leads to an enhanced local search algorithm for the LOP.

Theorem 1 *Let $M_{i,j} = \Delta Z(\pi, i, j)$ represent the gain of exchanging positions i and j on permutation π. Then,*

$$M_{i,j} = \begin{cases} -e^{\pi}_{i,j} & \text{when } i - j = 1, \\ -e^{\pi}_{i,j} + e^{\pi}_{i,j-1} + e^{\pi}_{i+1,j} & \text{when } i - j = 2, \\ -e^{\pi}_{i,j} + M_{i,j-1} + M_{i+1,j} - M_{i+1,j-1} + e^{\pi}_{i+1,j-1} & \text{when } i - j > 2, \end{cases}$$

where $e^{\pi}_{i,j} = e_{\pi(i)\pi(j)} - e_{\pi(j)\pi(i)}$.

Proof The first two cases are easy to check, so we consider only the third case. Using (14) and rearranging the terms, we have

$$M_{i,j} = -\sum_{k=i+1}^{j-2} e^{\pi}_{i,k} - \sum_{k=i+2}^{j-1} e^{\pi}_{k,j} - e^{\pi}_{i,j-1} - e^{\pi}_{i+1,j} - e^{\pi}_{i,j},$$

$$M_{i,j-1} = -\sum_{k=i+1}^{j-2} e^{\pi}_{i,k} - \sum_{k=i+2}^{j-2} e^{\pi}_{k,j-1} - e^{\pi}_{i+1,j-1} - e^{\pi}_{i,j-1},$$

$$M_{i+1,j} = -\sum_{k=i+2}^{j-2} e^{\pi}_{i+1,k} - \sum_{k=i+2}^{j-1} e^{\pi}_{k,j} - e^{\pi}_{i+1,j-1} - e^{\pi}_{i+1,j}, \quad \text{and}$$

$$M_{i+1,j-1} = -\sum_{k=i+2}^{j-2} e^{\pi}_{i+1,k} - \sum_{k=i+2}^{j-2} e^{\pi}_{k,j-1} - e^{\pi}_{i+1,j-1}.$$

Calculating the expression $M_{i,j-1} + M_{i,j-1} - M_{i+1,j-1} - M_{i,j}$ we are left with $e^{\pi}_{i,j} - e^{\pi}_{i+1,j-1}$, as desired. □

The insight to go from $\Delta Z(\pi, i, j)$ to a dynamic programming solution is to understand that the improvement for a pair of positions (i, j) can be decomposed into

Fig. 7 Local Search algorithm based on dynamic programming

```
procedure ImprovedLocalSearch()
01    max ← −∞
02    for k from 1 to n − 1
03        i ← 1
04        for j from k + 1 to n
05            delta ← e^π_{π(i)π(j)}
06            if j − i > 1 then
07                delta ← delta + M_{i,j−1} + M_{i+1,j}
08            end
09            if j − i > 2 then
00                delta ← delta − M_{i+1,j−1} − e^π_{π(i+1)π(j−1)}
11            end
12            M_{ij} ← delta
13            if delta > max
14                max ← delta; besti ← i; bestj ← j
15            end
16            i ← i + 1
17        end
18    end
19    return (max, besti, bestj)
end;
```

Fig. 8 The GRASP path-relinking procedure

```
procedure DoPathRelinking(EliteList, x)
1    TempSolution = ∅;
2    g = SelectSolutionAtRandom(EliteList);
3    while x ≠ g do
4        TempSolution = MakeNextMove(x, g);
5        LocalSearch(TempSolution);
6        if TempSolution is better than x or g then
7            EliteList = EliteList ∪ TempSolution;
8        AdjustEliteList(EliteList, ρ);
9    end;
end;
```

improvements for related pair of positions $(i+1, j)$, $(i, j-1)$, and $(i+1, j-1)$. If we represent by $M_{i,j}$ the variation in objective value when positions i and j are exchanged, then we can represent $M_{i,j}$ as the sum of $M_{i,j-1}$, $M_{i+1,j}$, and $-M_{i+1,j-1}$, along with the addition of e^{π}_{ij} and removal of $e_{i+1,j-1}$.

The local search algorithm now becomes a simple implementation of the result above, as shown in Fig. 7. The computational complexity of this algorithm is $O(n^2)$, since for each pair of positions, we are only using results that have been previously stored.

3.2.4 Path relinking

The solution provided by the local search procedure is used as the initial solution for the path-relinking. We randomly select a solution from the elite list as the guiding solution, determining the trajectory to be followed by the procedure. Figure 8 shows the pseudo-code for the path relinking procedure. The parameter ρ determines the size of the elite list as a fraction of the problem size.

With the trajectory defined by the two end solutions, we proceed to perform a series of moves that will transform the initial solution into the guiding solution. In each iteration, the algorithm performs a single move, thus creating a sequence of intermediate solutions (see Fig. 2). To add some greediness to the process, we search the 2-exchange neighborhood of the intermediate solutions. The solutions obtained in this manner are added to the elite list if they are better than either the initial or the guiding solutions. It should be noted that the search on the 2-exchange neighborhood may yield a previously examined solution in the path. However, this is not of concern in our implementation since we do not use the local minima during the procedure. The moving process terminates when the algorithm reaches the guiding solution. At this point, the elite list size could have grown considerably due to the added solutions. The procedure "AdjustEliteList" will discard the worst solutions, keeping the best ρn. The list is kept sorted at all times and therefore no sorting is needed.

4 Computational results

In this section we discuss the computational results we obtained when applying our algorithm to two sets of test problems:

1. *LOLIB.* These are real-world instances of linear ordering problems that are publicly available on the internet (Reinelt 2002). They consist of 49 input-output tables for some European countries, with sizes ranging from 44 to 60 objects.
2. *Mitchell.* This is a set of 30 random-generated instances by Mitchell (1997), with sizes ranging from 100 to 250 objects. Three different percentages of zero entries were used: 0, 10, and 20%, denoted by the last digit on the problem name. The generator as well as the instances are available at the author's web site (Mitchell 2002).

The optimal values for all instances are known. The generated instances are similar to those from LOLIB except for the numerical range of the entries—considerably larger for the latter. Despite attempts to replicate the characteristics of real-world instances such as those found in LOLIB, Mitchell's test set is significantly harder to solve. Most of the previous computational studies on the LOP mainly focus on the Mitchell instances, hence featuring only the LOLIB problems. The algorithm was written in C++ and executed on a Core Duo, 2.1 GHz, with 2 GB of memory. Empirically, we determined $\beta = 0.25$ and $\rho = 0.35$ as the best values for the parameters—e.g., for a problem of size n, the size of the RCL and the elite list are at most $0.25n$ and $0.35n$ respectively. The algorithm was executed five times for each problem instance, with a limit of 5000 GRASP iterations in its running time. We report the running time, number of iterations, and the gap between the best solution and the optimal solution. All times are reported in *seconds* and the gaps as *percentages*.

Table 1 shows the performance characteristics of our Revised GRASP with Path-Relinking algorithm tested on the LOLIB instances after 200 and 5000 iterations. These characteristics are compared with the one obtained by CPLEX and XPRESS. Note that CPLEX and XPRESS solves every LOLIB instance to optimality. The CPU times of CPLEX come from Belloni and Lucena (2004). The CPU times of XPRESS

Table 1 Performance characteristics of revised GRASP with path-relinking tested on the LOLIB instances after 200 and 5000 iterations versus the performance characteristics of CPLEX and XPRESS

Instance	Size	200 Iterations		5000 Iterations		CPLEX	XPRESS
		Gap (%)	Time (s)	Gap (%)	Time (s)	Time (s)	Time (s)
be75eec	50	0.0018	0.0310	0.00057	5.1410	1.83	2.61
be75np	50	0.0008	0.0001	0.00055	8.1250	74.68	7.44
be75oi	50	0.0017	0.2030	0.00080	3.6250	2.43	3.13
be75tot	50	0.0016	0.0160	0.00058	7.6410	1.84	2.70
stabu1	60	0.0023	0.2810	0.00161	16.3130	8.57	12.39
stabu2	60	0.0016	0.2970	0.00160	0.2970	5.69	9.94
stabu3	60	0.0038	0.3440	0.00119	11.3130	6.12	10.56
t59b11xx	44	0.0010	0.1880	0.00016	2.8440	1.93	1.97
t59d11xx	44	0.0012	0.0310	0.00002	1.7180	1.17	1.67
t59f11xx	44	0.0009	0.1410	0.00000	0.6410	0.99	1.59
t59i11xx	44	0.0000	0.0780	0.00000	2.8440	0.99	1.63
t59n11xx	44	0.0001	0.0310	0.00000	1.4380	1.25	1.58
t65b11xx	44	0.0014	0.0780	0.00039	1.4690	1.02	1.64
t65d11xx	44	0.0021	0.0940	0.00034	0.3600	1.13	1.61
t65f11xx	44	0.0019	0.0780	0.00078	1.0000	0.98	1.61
t65i11xx	44	0.0014	0.1720	0.00030	4.2030	1.09	1.59
t65l11xx	44	0.0001	0.0310	0.00000	1.0470	0.80	1.59
t65n11xx	44	0.0027	0.0780	0.00090	1.8910	1.36	1.63
t65w11xx	44	0.0010	0.1880	0.00019	2.0780	1.00	1.59
t69r11xx	44	0.0008	0.2030	0.00003	2.6250	1.19	1.59
t70b11xx	44	0.0011	0.1560	0.00018	2.0470	0.98	1.67
t70d11xn	44	0.0010	0.1880	0.00019	2.0470	24.17	2.20
t70d11xx	44	0.0017	0.0940	0.00015	5.3440	1.21	1.63
t70f11xx	44	0.0015	0.2030	0.00012	2.7500	1.01	1.59
t70i11xx	44	0.0020	0.0930	0.00076	3.7970	1.54	1.59
t70k11xx	44	0.0000	0.0001	0.00003	5.2970	1.02	1.64
t70l11xx	44	0.0004	0.0940	0.00000	0.2660	0.94	1.59
t70n11xx	44	0.0004	0.0150	0.00000	1.4380	1.06	1.63
t70u11xx	44	0.0015	0.2030	0.00037	0.9380	1.09	1.63
t70w11xx	44	0.0002	0.2350	0.00018	0.2190	1.01	1.59
t70x11xx	44	0.0004	0.1250	0.00010	4.0310	0.99	1.59
t74d11xx	44	0.0020	0.0630	0.00055	4.3440	1.20	1.59
t75d11xx	44	0.0004	0.0310	0.00022	5.6560	1.31	1.69
t75e11xx	44	0.0027	0.2030	0.00106	0.9060	1.00	1.61
t75i11xx	44	0.0002	0.1410	0.00011	4.0310	1.27	1.61
t75k11xx	44	0.0021	0.1560	0.00104	2.0160	1.24	1.61
t75n11xx	44	0.0007	0.1250	0.00002	0.8440	1.07	1.63
t75u11xx	44	0.0008	0.2810	0.00047	10.7810	1.10	1.58
tiw56n54	56	0.0008	0.2810	0.00047	10.7810	3.61	4.36
tiw56n58	56	0.0008	0.3280	0.00031	12.4380	3.25	4.27

Table 1 (*continued*)

Instance	Size	200 Iterations		5000 Iterations		CPLEX	XPRESS
		Gap (%)	Time (s)	Gap (%)	Time (s)	Time (s)	Time (s)
tiw56n62	56	0.0009	0.0470	0.00047	15.3750	3.67	4.39
tiw56n66	56	0.0018	0.3750	0.00091	5.5780	3.18	4.33
tiw56n67	56	0.0014	0.0160	0.00030	11.3600	4.67	4.64
tiw56n72	56	0.0012	0.5160	0.00032	7.9210	3.00	4.50
tiw56r54	56	0.0008	0.5160	0.00048	7.9220	3.28	4.42
tiw56r58	56	0.0014	0.2190	0.00040	14.9840	2.85	4.52
tiw56r66	56	0.0014	0.1560	0.00092	14.8600	2.79	4.28
tiw56r67	56	0.0014	0.4060	0.00090	13.1880	3.25	5.22
tiw56r72	56	0.0026	0.2190	0.00068	1.5000	2.96	4.74
Average		0.0013	0.1642	0.00044	5.08718	3.99	3.05
Std. Dev.		0.0008	0.1271	0.00041	4.69538	1.56	0.36

come from our experiments using Xpress–Mosel under Windows environment based on default settings. Table 2 shows the performance characteristics of our Revised GRASP with Path-Relinking algorithm tested on the Mitchell instances after 200 and 5000 iterations. Note that we do not report the performance characteristics of CPLEX and XPRESS tested on the Mitchell instances because they failed to solve the test problems within the time limit. The algorithm found optimal solutions for 47 out of 49 LOLIB instances, 17 of which were consistently solved to optimality. The average gap for the remaining LOLIB instances was 0.0013%. The average running time for these real-world instances was 0.16 seconds. Although none of the Mitchell instances were solved to optimality, the average gap after 5000 iterations was 0.001% with an average running time of 14.30 seconds. Table 3 summarizes the performance characteristics of our algorithm on the two test sets after 200 and 5000 iterations.

Table 4 compares the performance characteristics of our algorithm with other algorithms in the literature. $CK10$ represents the Chanas–Kobylański algorithm reported in Chanas and Kobylański (1996). represents the Tabu Search algorithm reported in Campos et al. (2001). SS and $SS10$ are two Scatter Search algorithms proposed in Campos et al. (1998). SS^2 and TS are the Context-Independent Scatter and Tabu Search algorithms for permutation problems recently proposed in Campos et al. (2005). PC and ND are the Position Cut and Node Degree procedures, respectively, proposed in the framework of Lagrangian heuristics for the LOP (Belloni and Lucena 2004). From the comparison, the results suggest that our algorithm seems to be the algorithm with the best appropriate trade-off between solution quality and CPU time. Figures 9 and 10 show the evolution of the gap as a function of the running time and the number of iterations for the LOLIB and Mitchell instances, respectively. Note that the units on the ordinates are percentage points. Tables 1 and 2 show the elapsed running time and gap values after 200 and 5000 iterations. The results reported are the averages of 5 runs for each problem instance. The averages of the values presented in these tables are shown in Table 3.

Table 2 Performance characteristics of revised GRASP with Path-Relinking tested on the Mitchell Instances after 200 and 5000 iterations

Instance	200 Iterations		5000 Iterations	
	Gap (%)	Time (s)	Gap (%)	Time (s)
r100a2	0.0268	0.2350	0.00191	58.6100
r100b2	0.0462	0.3600	0.00217	0.3750
r100c2	0.0416	0.0310	0.00180	19.2350
r100d2	0.0389	1.9220	0.00235	44.7970
r100e2	0.0270	1.9060	0.00257	51.7650
r150a0	0.0137	8.9690	0.00055	177.4530
r150a1	0.0263	11.6560	0.00110	155.7030
r150b0	0.0230	3.7970	0.00076	350.6720
r150b1	0.0224	10.4380	0.00118	234.6720
r150c0	0.0207	8.4060	0.00049	252.5940
r150c1	0.0369	4.9850	0.00118	167.5310
r150d0	0.0146	0.2500	0.00079	107.2660
r150d1	0.0212	6.7180	0.00095	338.5630
r150e0	0.0148	1.0780	0.00038	242.4690
r150e1	0.0388	10.7030	0.00133	141.9850
r200a0	0.0181	6.1100	0.00043	6.2970
r200a1	0.0409	6.2500	0.00102	546.5310
r200b0	0.0196	19.2350	0.00063	404.5160
r200b1	0.0333	12.7660	0.00101	785.6250
r200c0	0.0096	24.8910	0.00042	296.7660
r200c1	0.0244	27.5630	0.00109	212.7350
r200d0	0.0159	25.9060	0.00050	26.5000
r200d1	0.0345	20.5310	0.00108	665.7030
r200e0	0.0212	15.5630	0.00050	858.8750
r200e1	0.0297	0.3280	0.00086	43.4220
r250a0	0.0233	68.8130	0.00070	1565.1410
r250b0	0.0189	15.2030	0.00061	458.0940
r250c0	0.0159	3.2660	0.00064	3.3280
r250d0	0.0184	61.6100	0.00044	1133.6410
r250e0	0.0241	49.6100	0.00059	585.5310
Average	0.001344	14.3033	0.001002	331.2132
Std. Dev.	0.00082	17.6397	0.000601	368.2197

5 Conclusions

In this paper, we proposed and implemented a new heuristic based on a revised GRASP with path-relinking to efficiently obtain good solutions to linear ordering problems. The results demonstrated the benefit of embedding GRASP with a path-relinking procedure. The algorithm was tested on two sets of problems, exhibiting a remarkably robust performance as shown in Table 3. For all instances we obtained optimality gaps of less than 0.05% within 200 iterations and times ranging from 0.16

Table 3 Summary of the performance characteristics of revised GRASP with Path-Relinking obtained for the LOLIB and Mitchell test sets after 200 and 5000 iterations

Problem Set	Measure	200 Iterations		5000 Iterations	
		Gap (%)	Time (s)	Gap (%)	Time (s)
LOLIB	Average	0.0013	0.16	0.0004	5.08
	Std. Dev.	0.0008	0.12	0.0004	4.69
	Maximum	0.004	0.51	0.002	16.31
Mitchell	Average	0.001	14.30	0.001	331.21
	Std. Dev.	0.001	17.63	0.001	368.22
	Maximum	0.003	68.81	0.003	1565.1

Table 4 Results Comparison on the LOLIB Instances

Measure	GRASP-PR	CK10	SS	SS10	TS	SS^2	PC	ND
Gap (%)	0.002	0.150	0.010	0.010	0.001	0.003	0.004	0.000
Time (s)	0.869	0.100	2.350	14.280	12.800	8.900	8.254	6.157

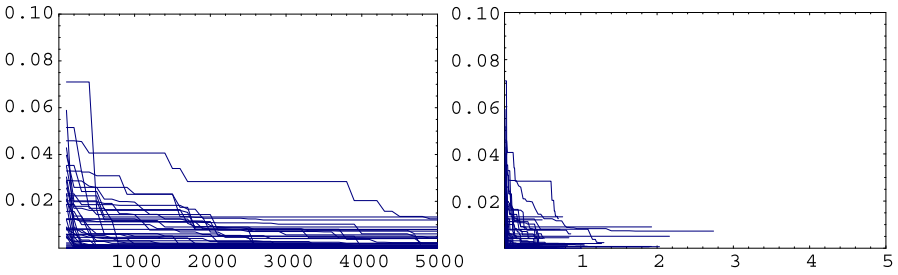

Fig. 9 LOLIB Instances: the gap from the optimal solution as a percentage is shown as a function of the number of iterations (*left*) and time (*right*). Time is in seconds

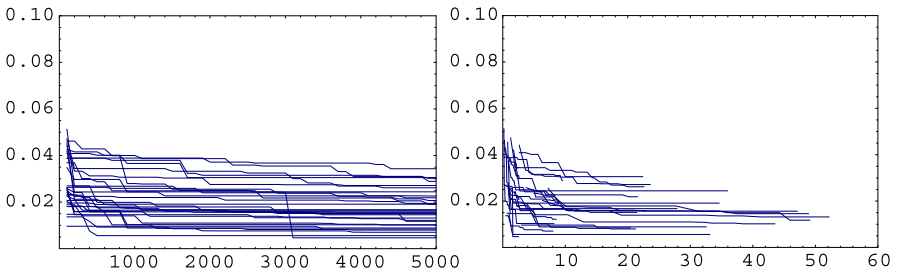

Fig. 10 Mitchell Instances: the gap from the optimal solution as a percentage is shown as a function of the number of iterations (*left*) and time (*right*). Time is in seconds

to 14.3 seconds on the average. We found optimal solutions for most of the LOLIB instances. The average gap for the Mitchell instances was 0.001%. With application in economics, researchers often use simulations in which many triangulation problems

need to be solved in limited time. The efficiency and high-quality performance of our algorithm makes it superior candidate for such applications. Nevertheless, we believe that several features can be added to improve the algorithm's long-run performance as well as further studying the effects of the parameters. GRASP algorithms, as well as most multi-start methods, do not generally require significant memory resources. However, added features such as path-relinking might compromise its performance, requiring a balance between intensification and efficiency. We have not observed such limitations in our implementation and thus we were able to fully exploit the intensification provided by the path-relinking method. More importantly, we applaud the work by Campos et al. (2005) on the development of a context-independent approach for permutation problems. We hope to develop GRASP with Path-Relinking in a context-independent manner as well.

Acknowledgements This work was partially supported by the NSF CAREER grant CCF–0546574 and Rutgers Research Council grant 202018.

References

Belloni A, Lucena A (2004) Lagrangian heuristics for the linear ordering problem, pp 37–63

Bolotashvili G, Kovalev M, Girlich E (1999) New facets of the linear ordering polytope. SIAM J Discrete Math 12:326–336

Burkard RE, Çela E, Pardalos PM, Pitsoulis LS (1998) The quadratic assignment problem. In: Pardalos PM, Du D-Z (eds) Handbook of combinatorial optimization. Kluwer Academic, Dordrecht, pp 241–338

Campos V, Laguna M, Martí R (1998) Scatter search for the linear ordering problem. Tech rep, Graduate School of Business, University of Colorado, Boulder, CO 80309, USA

Campos V, Glover F, Laguna M, Martí R (2001) An experimental evaluation of a scatter search for the linear ordering problem. J Glob Optim 21:397–414

Campos V, Laguna M, Martí R (2005) Context-independent scatter and tabu search for permutation problems. INFORMS J Comput 17:111–122

Chanas S, Kobylański P (1996) A new heuristic algorithm solving the linear ordering problem. Comput Optim Appl 6:191–205

Chaovalitwongse W, Kim D-K, Pardalos PM (2003) Grasp with a new local search scheme for vehicle routing problems with time windows. J Combin Optim 7:179–207

Chenery HB, Watanabe T (1958) International comparisons of the structure of production. Econometrica 26:487–521

Chiarini B, Chaovalitwongse W, Pardalos PM (2004) A new algorithm for the triangulation of input-output tables. In: Migdalas A, Pardalos PM, Baourakis G (eds) Supply chain and finance. World Scientific, Singapore, pp 254–273

Christof T, Reinelt G (1997) Low-dimensional linear ordering polytopes

Even G, Naor J, Rao S, Schieber B (2000) Divide-and-conquer approximation algorithms via spreading metrics. J ACM 47:585–616

Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 2:1–27

Festa P, Resende MGC (2000) GRASP: An annotated bibliography. Tech rep, AT&T Labs Research, Florham Park, NJ 07733, USA

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York

Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Dordrecht

Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J ACM 42:1115–1145

González CG, Pérez-Brito D (2001) A variable neighborhood search for solving the linear ordering problem. In: Proceedings of the MIC'2001-4th metaheuristics international conference, pp 181–185

Greistorfer P (2004) Experimental pool design: input, output and combination strategies for scatter search, pp 279–300

Grötschel M, Jünger M, Reinelt G (1984) A cutting plane algorithm for the linear ordering problem. Oper Res 2:1195–1220
Grötschel M, Jünger M, Reinelt G (1985) Facets of the linear ordering polytope. Math Programm 33:43–60
Hansen M (1989) Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems. In: Proceedings of the 30th annual symposium on foundations of computer science. IEEE Comput Soc, Los Alamitos, pp 604–609
Huang G, Lim A (2003) Designing a hybrid genetic algorithm for the linear ordering problem. In: Cantú-Paz E, Foster JA, Deb K et al (eds) Lecture notes in computer science, vol 2723. Springer, Berlin, pp 1053–1064
Jünger M (1985) Polyhedral combinatorics and the acyclic subdigraph problem. Research and Exposition in Mathematics, vol 7. Heldermann, Berlin
Laguna M, Martí R (1998) GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J Comput 11:44–52
Laguna M, Martí R, Campos V (1999) Intensification and diversification with elite tabu search solutions for the linear ordering problem. Comput Oper Res 26:1217–1230
Lee G, Lo S-C, Chen ALP (2002) Data allocation on wireless broadcast channels for efficient query processing. IEEE Trans Comput 51:1237–1252
Leontief W (1986) Input-output economics. Oxford University Press, London
Leung J, Lee J (1994) More facets from fences for linear ordering and acyclic subgraphs polytopes. Discrete Appl Math 50:185–200
Martí R (2003) Multi-start methods. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. International series in operations research & management sciences. Kluwer Academic, Dordrecht, pp 355–368, Chap 12
Mitchell JE (1997) Computational experience with an interior point cutting plane algorithm. Tech rep, Mathematical Sciences, Rensellaer Polytechnic Institute, Troy, NY 12180-3590, USA
Mitchell JE (2002) Generating linear ordering problems. http://www.rpi.edu/~mitchj/generators/linord
Mitchell JE, Borchers B (2000) Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In: Frenk H et al (eds) High performance optimization. Kluwer Academic, Dordrecht, pp 345–366, Chap 14. http://www.rpi.edu/mitchj/papers/combined.html
Newman A (2000) Approximating the maximum acyclic subgraph. Master's thesis, Massachusetts Institute of Technology
Newman A (2004) Cuts and orderings: On semidefinite relaxations for the linear ordering problem. In: Jansen K, Khanna S, Rolim JDP, Ron D (eds) Lecture notes in computer science, vol 3122. Springer, Berlin, pp 195–206
Newman A, Vempala S (2001) Fences are futile: on relaxations for the linear ordering problem. In: Proceedings of the eighth conference on integer programming and combinatorial optimization (IPCO), pp 333–347
Rao S, Richa AW (2004) New approximation techniques for some linear ordering problems. SIAM J Comput 34:388–404
Reinelt G (1985) The linear ordering problem: algorithms and applications. Research and exposition in mathematics, vol 8. Heldermann, Berlin
Reinelt G (1993) A note on small linear ordering polytope. Discrete Comput Geom 10:67–78
Reinelt G (2002) Linear ordering library (LOLIB). http://www.iwr.uni-heildelberg.de/iwr/comopt/soft/LOLIB/LOLIB.html