



The linear ordering problem with clusters: a new partial ranking

Javier Alcaraz¹ · Eva M. García-Nové¹ · Mercedes Landete¹ · Juan F. Monge¹

Received: 21 January 2020 / Accepted: 29 February 2020
© Sociedad de Estadística e Investigación Operativa 2020

Abstract

The linear ordering problem is among core problems in combinatorial optimization. There is a squared non-negative matrix and the goal is to find the permutation of rows and columns which maximizes the sum of superdiagonal values. In this paper, we consider that columns of the matrix belong to different clusters and that the goal is to order the clusters. We introduce a new approach for the case when exactly one representative is chosen from each cluster. The new problem is called the linear ordering problem with clusters and consists of both choosing a representative for each cluster and a permutation of these representatives, so that the sum of superdiagonal values of the sub-matrix induced by the representatives is maximized. A combinatorial linear model for the linear ordering problem with clusters is given, and eventually, a hybrid metaheuristic is carefully designed and developed. Computational results illustrate the performance of the model as well as the effectiveness of the metaheuristic.

Keywords Linear ordering problem · Rank aggregation problem · Bucket ordering problem · Metaheuristics

Mathematics Subject Classification 90C27 · 90C59 · 90C05

✉ Mercedes Landete
landete@umh.es

Javier Alcaraz
jalcaraz@umh.es

Eva M. García-Nové
eva.garcian@ua.es

Juan F. Monge
monge@umh.es

¹ Carr. de San Vicente del Raspeig, University of Alicante, San Vicente del Raspeig, 03690 Alicante, Spain

1 Introduction

The linear ordering problem is a well-established combinatorial optimization problem. Given a squared non-negative matrix, the problem consists in finding the permutation of rows and columns that maximizes the sum of superdiagonal values. The linear ordering problem has been studied in many fields including anthropology (Glover et al. 1974), machine translation (Tromble and Eisner 2009), and voting theory (Kemeny 1959). It is also related to the single-machine scheduling problem, where the objective is to minimize the total weighted (or average weighted) completion time (Grötschel et al. 1984). Given the broad interest in the problem, different computational strategies either exact or heuristics and metaheuristics have been developed. The book Marti and Reinelt (2011) provides a comprehensive study of the linear ordering problem and a thorough discussion of different solution techniques.

When the goal is to obtain the closest permutation of rows and columns to a given set of permutations and the distance among permutations is measured with the Kendall–Tau distance,¹ then the problem results in the rank aggregation problem (see Andoni et al. 2008; Dwork et al. 2001; Fagin et al. 2003; Yasutake et al. 2012 for a description and evolution of the rank aggregation problem and its applications). These two problems are closely related and it can be proved that the rank aggregation problem reduces to the linear ordering problem under a suitable transformation (García-Nové et al. 2017).

Both of the above problems, the linear ordering problem and the rank aggregation problem give total orders, i.e., do not leave unordered columns. If only a partial order of the columns/rows of a squared matrix is required, the solution would be a *bucket order* instead of a permutation. A bucket order is an ordered partition of the columns into *buckets*, so that all the elements (columns/rows) within a bucket are assumed to be tied or incomparable and the order between two elements of different buckets is given by the relative ordering of the buckets which they belong to. The notion of bucket order was formalized by Fagin et al. (2004) as a way to approach the rank aggregation problem with ties. Given a squared matrix with all the entries in $[0, 1]$, the bucket order problem (Fagin et al. 2003, 2004; Feng et al. 2008; Gionis et al. 2006) consists of computing the bucket order that best captures the data. The bucket order problem has been used to discover ordering information among elements in many applications. It is used in the context of seriation problems in scientific disciplines such as Paleontology (Fortelius et al. 2006; Puolamaki et al. 2006), Archaeology (Halekoh and Vach 2004), and Ecology (Miklos et al. 2005), and also to aggregate browsing patterns of visitors to a web portal (Feng et al. 2008). The bucket order problem gives a clustering of columns (rows).

In this paper, we consider that columns of the matrix belong to different clusters and that the goal is to obtain *good* permutations of the clusters. The goal is to choose a representative column of each cluster and an order of the representative

¹ The Kendall–Tau distance (Kendall 1938) is a classical measure for comparing two permutations which counts the number of pairs for which the order is different in both permutations.

columns, such that the sum of superdiagonal values of the sub-matrix induced by the representatives is maximum. Contrary to the bucket order problem, the column affiliation with clusters is an input instead of an output. If columns represent politicians, clusters could be parties; if columns represent athletes, clusters could be teams, etc. We propose a new approach to choose a representative of each cluster and find the best permutation of these representatives. The new problem is appropriated when scheduling tasks, such that only one task of each cluster is required. Also when a representative of each cluster is required. Some Spanish Universities give a set of rewards, one for each Department, among all the teachers that sign up for the National Teaching Evaluation Program called Docencia (<https://programadocencia.umh.es/>), and then, the Departments can be sorted according to the scores awarded to their teachers.

The new ranking order introduced in this paper generalizes the linear ordering problem and the rank aggregation problem. In fact, the linear ordering problem and the rank aggregation problem have been proved to be the same in some particular cases. In the case where clusters have one element, the new ranking orders coincide with linear ordering and rank aggregation. Besides, the new ranking orders are alternative partial rankings different from the bucket order problem and useful when the clusters are known and when the belonging of an element to a cluster is not part of the solution but of the data themselves.

Clusters are different to buckets, because clusters are inherent to the population and thus are inputs of the problem, while buckets are induced by the data in entries in the matrix and are outputs of the problem.

A small example is kept throughout the paper which illustrates the different ranking orders as well as the different percentages of voter preferences that can be achieved. In fact, it is the example proposed in Feng et al. (2008) for illustrating the bucket order problem.

The remainder of this paper is organized as follows. In Sect. 2, we give some preliminaries and definitions. In Sect. 3, we model the new ranking problem and we discuss about its integrality and linear gaps as well as about how to adapt valid inequalities from the Linear Ordering Problem. In Sect. 4, we propose a hybrid metaheuristic algorithm for the Linear Ordering Problem with Clusters. This will be followed in Sect. 5 by extensive numerical analysis, showing the performance of the linear ordering problem with clusters model and of the metaheuristic algorithm. Conclusions are drawn in Sect. 6.

2 Preliminaries and notation

In this section, the definition of two known ranking methods is revised, the linear ordering problem and the rank aggregation problem, both consisting of obtaining total rankings.

Let $V = \{1, \dots, n\}$ be a set of indices and let C be a non-negative $n \times n$ matrix. Let R be the set of all the permutations of V . The Linear Ordering Problem (LOP) for matrix C consists of finding the permutation $\pi \in R$, such that

Table 1 Candidate rankings: list of five total orders

Voter 1	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Voter 2	<i>a</i>	<i>c</i>	<i>b</i>	<i>d</i>	<i>f</i>	<i>e</i>
Voter 3	<i>a</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>f</i>	<i>e</i>
Voter 4	<i>d</i>	<i>f</i>	<i>c</i>	<i>a</i>	<i>b</i>	<i>e</i>
Voter 5	<i>c</i>	<i>f</i>	<i>e</i>	<i>d</i>	<i>b</i>	<i>a</i>

Table 2 Matrix *C* of preferences

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
<i>a</i>		0.8	0.6	0.6	0.8	0.6
<i>b</i>	0.2		0.2	0.4	0.8	0.6
<i>c</i>	0.4	0.8		0.8	1.0	0.8
<i>d</i>	0.4	0.6	0.2		0.8	0.8
<i>e</i>	0.2	0.2	0.0	0.2		0.2
<i>f</i>	0.4	0.4	0.2	0.2	0.8	

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{\pi(i)\pi(j)}$$

is maximized.

Let $R' \subseteq R$ be a set of permutations of V , and let $d(r_1, r_2)$ be the Kendall–Tau distance between r_1 and r_2 for all $r_1, r_2 \in R'$. The Rank Aggregation Problem (RAP) consists of finding the permutation $\pi \in R$, such that:

$$\sum_{r \in R'} d(\pi, r)$$

is minimized.

Both LOP and RAP are defined for general non-negative squared matrices. Anyway, in practice, general non-negative rankings can be transformed in $[0, 1]$ ranking in several ways.

The following example illustrates the kind of solution of the LOP.

Example 1 Let $V = \{a, b, c, d, e, f\}$ be a set of six candidates. Suppose that five voters rank them as Table 1 shows. The table corresponds to an example in Feng et al. (2008). Thus, the square non-negative matrix C in Table 2 represents the preferences: c_{ij} is the fraction of rankings in which candidate i is listed (ranked) before candidate j .

The solution for Linear Ordering Problem (LOP), which maximizes the preferences of the voters on the 6 candidates, is the rank ($a | c | d | b | f | e$) with an optimal value of 11.2. If a consensus permutation with zero disagreements with respect to a database of permutations was to exist, the optimal value of LOP would be 15, the superdiagonal sum of ones; therefore, 74.6% ($= (11.2/15) \times 100$) is the percentage in which the linear order solution guarantees voter preferences.

3 A new partial ranking

In this section, a new partial ranking is introduced for the case where V is partitioned into disjoint subsets (clusters): $V = V_1 \cup V_2 \dots V_m$ and $V_r \cap V_s = \emptyset$ for all $r, s \in \{1, \dots, m\} r \neq s$.

We define $M = \{1, \dots, m\}$.

3.1 The linear ordering problem with clusters

The Linear Ordering Problem with Clusters (LOP-C) consists in finding the best linear order for V_1, \dots, V_m when only one representative of each cluster is considered to define the order.

For all $i \in V$, let z_i be a binary variable which takes the value of one, if and only if index i is the representative of its cluster. For all $i, j \in V$ belonging to different clusters, let x_{ij} be a binary variable which takes the value of one if and only if indexes i and j are the representative of their clusters and the cluster to which i belongs to goes before the cluster to which j belongs to. The IP formulation of the LOP-C can be stated as follows:

$$(LOP-C) \max \sum_{r,s \in M: r \neq s} \sum_{\substack{i \in V_r \\ j \in V_s}} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{\substack{k \in V_t \\ j \in V_s}} x_{kj} - \sum_{\substack{i \in V_r \\ j \in V_s}} x_{ij} - \sum_{\substack{k \in V_t \\ i \in V_r}} x_{ki} \leq 0 \quad r, s, t \in M: \text{pairwise disjoint} \tag{2}$$

$$\sum_{j \in V_r} (x_{ij} + x_{ji}) - z_i = 0 \quad r \in M, i \notin V_r \tag{3}$$

$$\sum_{i \in V_r} z_i = 1 \quad r \in M \tag{4}$$

$$x_{ij} \in \{0, 1\} \quad i \in V_r, j \in V_s, r \neq s \tag{5}$$

$$z_i \in \{0, 1\} \quad i \in V, \tag{6}$$

The objective function (1) is the weighted sum of all the x -variables. Even if matrix C has $|V| \times |V|$ elements, only the elements in positions associated with indexes belonging to different clusters are required. Constraints (2) state transitivity in the order of clusters, if cluster t goes before cluster s and cluster s goes before cluster r , then cluster t must go before cluster r . Constraints (3) state that given i the representative of a cluster and another cluster V_r , there is a variable x_{ij} or a variable x_{ji} with $j \in V_r$ which takes the value of one and $x_{jk} = 0$ if $j, k \neq i$. Constraints (4) entail

Table 3 Voter preferences about selected candidates

Voter 1	<i>b</i>	<i>c</i>	<i>e</i>
Voter 2	<i>c</i>	<i>b</i>	<i>e</i>
Voter 3	<i>c</i>	<i>b</i>	<i>e</i>
Voter 4	<i>c</i>	<i>b</i>	<i>e</i>
Voter 5	<i>c</i>	<i>e</i>	<i>b</i>

that only one element from each cluster is selected. Constraints (5) and (6) are the domain constraints.

Any feasible solution of LOP-C induces a permutation of m elements in V , one in each cluster and thus a cluster permutation. The indexes of the variables in LOP-C are in V , while the optimal solution is an order of the elements of M .

Example 1 (cont) Let $V_1 = \{a, b\}$, $V_2 = \{c, d\}$ and $V_3 = \{e, f\}$. The optimal solution for the LOP-C is $(c, b, e) \equiv (\mathbf{c}, d \mid a, \mathbf{b} \mid \mathbf{e}, f)$ and the optimal value is 2.6. In terms of variables, $z_c = z_b = z_e = 1$, $x_{cb} = x_{ce} = x_{be} = 1$ and the rest of variables take the value zero. If the preferences of non-selected candidates are removed, the list of preferences would be the list in Table 3 and a consensus permutation with zero disagreement would give an objective value of 3. Then, the optimal solution (c, b, e) guarantees $86.6\%((2.6/3) \times 100)$ of voter preferences.

Besides, the LOP solution does not lead to the LOP-C solution. The LOP solution is $(a \mid c \mid d \mid b \mid f \mid e)$, then the first three elements of different clusters are $(a, c, f) \equiv (\mathbf{a}, b \mid \mathbf{c}, d \mid \mathbf{e}, \mathbf{f})$, which is not the LOP-C solution. In fact, the objective value for the solution (a, c, f) in the LOP-C is 2, which only guarantees the $66.6\%((2/3) \times 100)$ of voter preferences.

The Linear Ordering Problem with Clusters (LOP-C) is NP-hard, since it is the Linear Ordering Problem (LOP) when all the clusters have a single element.

3.2 Integrality gap, linear gap, and valid inequalities

In this section, we show that LOP-C is more difficult than LOP in terms of integrality gap and thus in terms of linear gap. Furthermore, we adapt some well-known families of valid inequalities of the LOP to the LOP-C.

In Boussaid et al. (2013), the authors establish that the LOP is “asymptotically easy”. Under certain mild probability assumptions, the ratio between the best and worst solution is arbitrarily close to 1 with probability tending to 1 if the problem size goes to infinity. If all the elements of the preference matrix follow from a uniform distribution, all the *swapped* matrices look similar.

In Marti and Reinelt (2011), the authors define the integrality gap of the LOP as the ratio between the optimal value of the relaxation of the problem that allows in-transitivity and the integer optimal value. Note that when the transitivity constraint is removed, the relaxed LOP is trivial and its optimal value is achieved by forcing $x_{rs} = 1$ if $m_{rs} > m_{sr}$. Later, in Aparicio et al. (2019), the authors prove that

the integrality gap of LOP converges to $4/3$ if the preference matrix is in a certain normal form.

LOP-C is more difficult than LOP in the sense that the relaxed LOP-C is not trivial. The following example illustrates this issue.

Remark 3.1 The relaxation of the transitivity constraints in LOP-C does not provide an integer solution.

Example 2 Let $V = \{a, b, c, d, e, f\}$ be a set of six elements in three clusters $V_1 = \{a, b\}$, $V_2 = \{c, d\}$ and $V_3 = \{e, f\}$, and let the following matrix be the preference matrix.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
<i>a</i>			1	2	2	0
<i>b</i>			3	1	0	1
<i>c</i>	0	2			3	1
<i>d</i>	0	0			1	1
<i>e</i>	0	1	3	2		
<i>f</i>	1	2	0	3		

The optimal solution of LOP-C is $(e, b, c) \equiv (\mathbf{e}, f, | a, \mathbf{b} | c, d)$ and the optimal value is 7. If the transitivity constraints are relaxed, we obtain the same solution, but if the transitivity constraints are relaxed and the linear relaxation of the problem is considered, then the optimal value is 7.5 with a fractional solution, $z_i = 0.5 \forall i$, and $x_{14} = x_{23} = x_{15} = x_{62} = x_{53} = x_{64} = 0.5$.

The integrality gap is related with the linear gap in the sense that the integrality gap gives an upper bound of the linear gap: if the integrality gap of a LOP is α , its linear gap is smaller than $\alpha - 1$. Thus, the linear gap for LOP is expected to be small. In fact, in Marti and Reinelt (2011), the authors propose several families of valid inequalities for the LOP and conclude that the linear gap is scarcely reduced when these are added to the problem.

Since the LOP-C is more complex than the LOP gap-wise, valid inequalities for LOP-C could be sensible for reducing its linear gap. For instance, since $x_{ij} + x_{ji} = z_i z_j$ for all i, j , the valid inequalities for the Boolean polytope in Remark 3.2 and the equalities in Remark 3.3 apply.

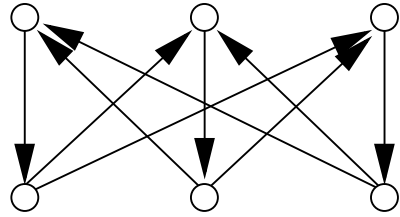
Remark 3.2 The following inequalities are valid for LOP-C:

$$z_i + z_j \leq 1 + x_{ij} + x_{ji} \quad i \in V, j \in V.$$

Remark 3.3 The following equalities are valid for LOP-C:

$$\sum_{i \in V} (x_{ij} + x_{ji}) = (m - 1)z_j \quad j \in V.$$

Fig. 1 3-fence graph



On the other hand, all the valid inequalities for LOP in the shape of $\sum_{(i,j) \in T} x_{ij} \leq t$ for $T \subseteq V \times V, t \in \mathbb{N}$ can become a valid inequality for LOP-C by replacing variables by sum of variables.

By way of example, the LOP 3-fence inequality:

$$x_{if} + x_{ff} + x_{fk} + x_{jg} + x_{gi} + x_{gk} + x_{kh} + x_{hi} + x_{hj} \leq 7;$$

f, g, h, i, j, k pairwise disjoint, which represents the 3-fence graph in Figure 1, becomes the inequality in Remark 3.4 for LOP-C.

Remark 3.4 The following 3-fence inequality is a valid inequality for LOP-C:

$$\begin{aligned} & \sum_{\substack{i \in V_r \\ f \in V_u}} x_{if} + \sum_{\substack{f \in V_u \\ j \in V_s}} x_{ff} + \sum_{\substack{f \in V_u \\ k \in V_t}} x_{fk} \\ & + \sum_{\substack{j \in V_s \\ g \in V_v}} x_{jg} + \sum_{\substack{g \in V_v \\ i \in V_r}} x_{gi} + \sum_{\substack{g \in V_v \\ k \in V_t}} x_{gk} \\ & + \sum_{\substack{k \in V_t \\ h \in V_w}} x_{kh} + \sum_{\substack{h \in V_w \\ i \in V_r}} x_{hi} + \sum_{\substack{h \in V_w \\ j \in V_s}} x_{hj} \leq 7 \\ & r, s, t, f, g, h \in M : \text{pairwise disjoint.} \end{aligned}$$

4 A hybrid metaheuristic

For the last decades, metaheuristic techniques have been imposed on simple heuristics to give approximate solutions to difficult combinatorial optimization problems. These techniques incorporate procedures that, well designed, are able to escape from local optima to achieve quality solutions. Scatter search (SS) and Genetic Algorithms (GA) are two of the most widely used and can be hybridized to improve their performance, both in terms of speed and effectiveness. SS was first introduced by Glover (1977) who described it as a method that uses a succession of coordinated initializations to generate solutions. The original proposal did not provide

certain implementation details and later Glover in (1994) provided such details and expanded the scope of application of the method to nonlinear, binary, and permutation problems. In Glover (1998), Glover gives the scatter search template. The basic idea of the method is to generate a systematically dispersed set of points from a chosen set of reference points to maintain a certain diversity level among the members of this set. Genetic algorithms were first introduced by Holland (1975) and imitate the evolution of species, based on the survival of the fittest principle. These algorithms maintain a population of solutions and apply a set of genetic operators like crossover or mutation to generate new individuals and maintain an appropriate level of diversity. The original proposals of these methods have been later transformed by several authors, incorporating advanced designs and procedures obtaining hybrid algorithms that have been successfully applied to several combinatorial optimization problems. A description of some of these methods and their applications are described in Boussaid et al. (2013) and Laguna and Marti (2003), among others.

Algorithm 1 Hybrid metaheuristic for LOP-C: MHLLOP-C

Step 0: (Initialization)

Generate_Initial_Population(*Pop_size*);

Generate_RefSet(*RefSet_size*);

Step 1: (Iterations)

while not (stopping criterion) **do**

Adapted_Path_relinking();

$\pi^0 := \text{Random_Select_RefSet}()$;

switch (*random_int*(1,4))

case 1:

$\pi^1 := \text{Interchange}(\text{Insertion}(\text{OneNodeMutation}(\pi^0)))$;

case 2:

$\pi^1 = \text{Interchange}(\text{Insertion}(\text{AllNodeMutation}(\pi^0)))$;

case 3:

$\pi^1 := \text{Insertion}(\text{Interchange}(\text{OneClusterMutation}(\pi^0)))$;

case 4:

$\pi^1 := \text{Insertion}(\text{Interchange}(\text{AllClusterMutation}(\pi^0)))$;

end switch

Update_RefSet(π^0, π^1);

end while

Step 2: (Final Improvement)

RefSet_Improvement();

In this paper, we propose an algorithm to give approximate solutions for the Linear Ordering Problem with Clusters (LOP-C) which is based on the Scatter Search template but introduces efficient procedures, some of them based on

genetic operators, that incorporate problem specific knowledge. This makes the proposed algorithm an efficient hybrid metaheuristic tool to manage this problem. The algorithm starts generating an initial population pool of randomized feasible solutions of a given size. Then, a reference set of solutions with a good level of quality and diversity is selected from the initial population. Later, the evolutive process starts and is repeated until a stopping criterion is satisfied. First, a procedure based on path relinking is applied over two solutions of the RefSet, creating a new solution that can replace one of the originals. Then, a new solution of this set is randomly chosen and undergoes three different operations: mutation, interchange, and insertion. The order in which these operators are applied over the solution depends on the type of mutation, which is selected randomly between two different types: one affects nodes and another modifies clusters. Moreover, each type has two different versions, depending on whether it affects only one or all the nodes/clusters of the solution, and the version to be applied is also chosen randomly. After these operations are carried out, RefSet is updated and the new solution replaces the original one in RefSet if it is better. Once the evolution process is finished, a final improvement procedure is performed to increase the quality of the solutions in RefSet. It consists of applying to every solution in RefSet, the interchange and insertion techniques in a successive way but in a random order. Once more, the modified solution replaces the original one if it has been improved. The result of the algorithm is the best solution of the reference set.

In the next subsections, the main operators and procedures performed by the hybrid metaheuristic are described in detail. In the following, we define the function $\rho : V \rightarrow M$ which indicates the cluster to which the column belongs: $\rho(i) = r$ iff $i \in V_r$. Abusing of notation, $\rho(\pi)$ shall represent the vector $(\rho(\pi(1)), \dots, \rho(\pi(m)))$. Moreover, we shall refer to the objective value in LOP-C of a feasible solution π as $v(\pi)$.

Example 1 (cont). Let $V = \{a, b, c, d, e, f\}$, $V_1 = \{a, b\}$, $V_2 = \{c, d\}$, $V_3 = \{e, f\}$. The optimal solution for the LOP-C is a node permutation $\pi^* = (c, b, e) \equiv (\mathbf{c}, \mathbf{d} \mid \mathbf{a}, \mathbf{b} \mid \mathbf{e}, \mathbf{f})$, associated with a cluster permutation $\rho(\pi^*) = (2, 1, 3)$.

4.1 Initial population and reference set

In general, the initial population (IniPob) is a set of feasible solutions that can be obtained either randomly or through a specific algorithm. The first mechanism is faster, but the quality of the solutions is poorer and the second has the advantage of creating good solutions, but requires more computational effort.

In the proposed algorithm, we have employed a pure random mechanism to generate an initial population of feasible solutions, IniPob, sized *Pop_size*. Each solution is a partial ranking of V given by a permutation π of length m . After creating the IniPob, a set of *RefSet_size* good and diverse solutions called Reference Set (RefSet) is chosen from IniPob.

The construction of the initial RefSet starts with the selection of the best *RefSet_size/2* solutions [best value of the objective function (1)] from the initial

population. The remaining $RefSet_size/2$ solutions are included in the RefSet to increase the level of diversity in it. The distance between two solutions $\pi^0 \in RefSet$ and $\pi^1 \in IniPob \setminus RefSet$ is defined as:

$$D(\pi^0, \pi^1) = \frac{\alpha}{2} \left(\frac{d(\rho(\pi^0), \rho(\pi^1))}{\frac{m(m-1)}{2}} + \frac{s}{m} \right) + (1 - \alpha) \left(\frac{v(\pi^1)}{v(\pi^0)} \right), \tag{7}$$

where α is a value in $(0, 1)$, $d(\rho(\pi^0), \rho(\pi^1))$ is the Kendall–Tau distance, s is the number of clusters with different representatives, and $m(m - 1)/2$ is the maximum value of a Kendall–Tau distance between two permutations of size m . Note that the distance function D is bounded by 1 when $\alpha \in (0, 1)$.

The minimum distance from each solution in IniPob to the solutions in RefSet is computed. Then, the solution with the maximum of these minimum distances is added to RefSet. This process is repeated $RefSet_size/2$ times. The resulting reference set has $RefSet_size/2$ high-quality solutions and $RefSet_size/2$ diverse solutions.

Example 1 (cont). Let $\pi^0 := (d | a | e) \equiv (c, \mathbf{d} | \mathbf{a}, b | \mathbf{e}, f)$ and $\pi^1 = (f | a | c) \equiv (e, \mathbf{f} | \mathbf{a}, b | \mathbf{c}, d)$. The distance between π^0 and π^1 is:

$$D(\pi^0, \pi^1) = \frac{\alpha}{2} \left(\frac{3}{\frac{3(3-1)}{2}} + \frac{2}{3} \right) + (1 - \alpha) \left(\frac{v(\pi^1)}{v(\pi^0)} \right).$$

4.2 Adapted path relinking

Path relinking was originally proposed by Glover (1996) as an intensification strategy to explore trajectories connecting elite solutions obtained by tabu search or scatter search (Glover and Laguna 1997; Glover et al. 2000, 2004). The path relinking mechanism produces new solutions combining every pair of solutions in the RefSet. Given one pair of solutions selected to undergo the mechanism, one is used as the origin of the path and the other as the end of the same path. Going from the origin to the end consists of generating a set of intermediate solutions. The resulting solution is the best of the path.

We have designed an adapted path relinking operator which randomly chooses only one pair of solutions of RefSet, among the $RefSet_size/2$ best solutions. The operator is bi-directional and moves from the origin to the end and then from the end to the origin. The operator evaluates all the neighbour solutions generated in both paths, returning the best as result. This solution replaces the worst of the originals if it is better. Given two solutions from RefSet, π^0 and π^1 , the way of moving from π^0 to π^1 is depicted in Algorithm 2.

Algorithm 2 Path between two solutions

Step 0: (initialization)

if $(v(\pi^0) > v(\pi^1))$ then

$\pi^* := \pi^0;$

else

$\pi^* := \pi^1;$

end if

$\pi^2 := \pi^0;$

$r = 1;$

Step 1: (Algorithm)

while $\pi^2 \neq \pi^1$ do

if $(\rho(\pi^2(r)) = \rho(\pi^1(r)))$ then

if $(\pi^2(r) = \pi^1(r))$ then

$r = r + 1;$

else

$\pi^2(r) := \pi^1(r);$

if $(v(\pi^2) > v(\pi^*))$ then

$\pi^* := \pi^2;$

end if

end if

else

$t = r;$

repeat

$t = t + 1;$

until $\rho(\pi^2(t)) = \rho(\pi^1(r))$

$\pi^3 := \pi^2;$

$\pi^2(r) = \pi^3(t);$

for $s = 1$ to $t - r$ do

$\pi^2(r + s) = \pi^3(r + s - 1);$

end for

if $(v(\pi^2) > v(\pi^*))$ then

$\pi^* := \pi^2;$

end if

end if

end while

Example 1 (cont). Let $\pi^0 = (a | d | e) \equiv (\mathbf{a}, b | c, \mathbf{d} | \mathbf{e}, f)$ and $\pi^1 = (c | b | e) \equiv (\mathbf{c}, d | a, \mathbf{b} | \mathbf{e}, f)$. The first neighbour solution is $\pi = (d | a | e) \equiv (c, \mathbf{d} | \mathbf{a}, b | \mathbf{e}, f)$, and the second is $\pi = (c | a | e) \equiv (\mathbf{c}, d | \mathbf{a}, b | \mathbf{e}, f)$, while the third and the last is $\pi = (c | b | e) \equiv (\mathbf{c}, d | a, \mathbf{b} | \mathbf{e}, f)$.

4.3 Insertion

Given a solution π and two positions $r, s \in M$, we define the insertion of element $\pi(r)$ in position s as a result of adding at position s the element $\pi(r)$ and moving all subsequent elements up one position. Given a solution, the insertion operator

evaluates all the possible insertions in that solution checking the objective function for each. The result of the insertion operator is the best of the evaluated solutions. This operator is based on the idea of the “adding” procedure used in the metaheuristic proposed in Alcaraz et al. (2019).

Remark 4.1 Let π^0 be a solution of LOP-C and let r, s in M . Let π be the result of inserting the element $\pi^0(r)$ in position s , and then:

$$\begin{aligned} \text{if } r < s \quad v(\pi) &= v(\pi^0) + \sum_{t=r+1}^s (c_{\pi^0(t)\pi^0(r)} - c_{\pi^0(r)\pi^0(t)}) \\ \text{if } r > s \quad v(\pi) &= v(\pi^0) + \sum_{t=s}^{r-1} (c_{\pi^0(r)\pi^0(t)} - c_{\pi^0(t)\pi^0(r)}). \end{aligned}$$

Remark 4.1 facilitates the evaluation of the new solutions. While the calculation of the objective function by (1) requires $m(m - 1)/2$ operations, the calculation by the formula in Remark 4.1 requires between 2 and $2(m - 1)$ operations, depending on the positions being considered.

4.4 Mutation

Mutation is used as one of the most important operators in GA, imitating the mutation of genetic material that sometimes occurs in nature, changing the characteristics of an individual. The mutation mechanism permits the introduction of new characteristics that were not present in the population into a solution or characteristics that some individuals had in the past but were lost in the evolution process. This is very important in evolutionary algorithms, to introduce variability and avoid being trapped in local optima, producing a premature convergence of the algorithm. We have designed a mutation strategy that, making use of the problem-knowledge, allows us to introduce diversity into the RefSet in a very appropriate way. Here, we present two different types of mutation, mutation of nodes and mutation of clusters, and for each one, two different versions depending on whether only one or all the nodes/clusters mutate. Given a solution π of the RefSet: OneNodeMutation(π), AllNodeMutation(π), OneClusterMutation(π), and AllClusterMutation(π) can be applied to it. The first two procedures mutate nodes, while the others produce the mutation of clusters.

Given a solution π in RefSet, OneNodeMutation(π) chooses a random node $i \in V$, such that i is a node of the solution π , that is $i = \pi(r)$, and it is replaced by a random node $j \in V_r$ with a probability of mutation, P_{mut1} ; AllNodeMutation(π) selects every node $i \in V$, such that i is a node of the solution π , that is $i = \pi(r)$, and it is replaced by a random node $j \in V_r$ with a probability of mutation, P_{mut2} ; OneClusterMutation(π) chooses two random clusters $r, s \in M$ and inserts the element $\pi(r)$ in the position of cluster s (OneClusterMutation works as in Sect. 4.3) with a given probability P_{mut1} ; Finally, AllClusterMutation(π) applies OneClusterMutation(π) for all $r \in M$, but with probability P_{mut2} for each cluster. Therefore, the result of this procedure could be the original solution or the mutated one.

Example 1 (cont). Let $\pi^0 = (a | d | e) \equiv (\mathbf{a}, b | c, \mathbf{d} | \mathbf{e}, f)$. The output for OneNodeMutation(π^0) with $i = c$ is $(a | c | e) \equiv (\mathbf{a}, b | \mathbf{c}, d | \mathbf{e}, f)$. The output for AllNodeMutation(π^0) with random values b, d, e is $(b | d | e) \equiv (a, \mathbf{b} | c, \mathbf{d} | \mathbf{e}, f)$. The output for OneClusterMutation(π^0) with $r = 3$ and $s = 1$ is $(e | a | d) \equiv (\mathbf{e}, f, | \mathbf{a}, b | c, \mathbf{d})$. Eventually, the output for AllClusterMutation(π^0) when the first cluster is selected to move to the third position, the second cluster is selected to move to the third position, and the third cluster is selected to move to the first position is $(e | d | a) \equiv (\mathbf{e}, f | c, \mathbf{d}, | \mathbf{a}, b)$.

4.5 Interchange

The interchange method was introduced by Teitz and Bart (1968) and consists of interchanging one solution attribute that is in the solution with one that is not. An extension of this version is the so-called k-interchange, in which k solution attributes are interchanged (Mladenovic et al. 1996).

We use a particular m-interchange previously proposed as operator of a metaheuristic algorithm for a location problem (Alcaraz et al. 2012). Given a solution π , this operator replaces each element i in π for the best element $i^* \in V_{\rho(i)}$. The best element i^* is the one whose interchange leads to the best possible objective value.

To evaluate the objective value for each $j \in V_{\rho(i)}$, it is useful to make the following remark.

Remark 4.2 Let π^0 and π be two solutions of LOP-C, such that $\rho(\pi^0(r)) = \rho(\pi(r))$, for all $r \in M$, and $\pi^0(s) \neq \pi(s)$ for exactly one $s \in M$. Then:

$$v(\pi) = v(\pi^0) + \sum_{s=1}^{r-1} (c_{\pi^0(s)\pi(r)} - c_{\pi^0(s)\pi^0(r)}) + \sum_{s=r+1}^m (c_{\pi(r)\pi^0(s)} - c_{\pi^0(r)\pi^0(s)}).$$

Remark 4.2 also facilitates the evaluation of the new solutions. Again, the calculation of the objective function by (1) requires $m(m - 1)/2$ operations, while the calculations by the formula in Remark 4.2 requires $2(m - 1)$.

The resulting solution of this procedure replaces the original solution in the RefSet only if it a better objective value.

5 Computational study

We have used a set of instances in <http://www.optiscom.es/lolib/> for the Linear Ordering. In particular, the set of instances comprises the 25 instances of size 100 in the group of problems called *Random instances of type AI* which are generated from a [0, 100] uniform distribution and were proposed in Reinelt (1985) and generated in Campos et al. (2001). Each instance of size $n = 100$ has been split in $m = 4, 10, 20, 50$ clusters of equal size. Thus, we have used 100 different problems that form 4 groups.

All tests were performed on a PC with a 2.33 GHz Intel Xeon dual core processor, 8.5 GB of RAM, and LINUX Debian 4.0 operating system. A CPLEX v.11.0

optimization engine was used for solving the LOP-C model, while MH_LOP-C was implemented in C.

First, we have solved the model LOP-C for all the instances imposing a time limit of 3 h and for each group of instances, $m = 4, 10, 20, 50$. After that, we have solved the linear relaxation and calculated the linear Gap of each instance.

Table 4 report the results for $m = 4, 10, 20, 50$. Headings are as follows. The first column is the instance number. The second, third, fourth, and fifth blocks of columns are the results for the different clusters: v^* shows the objective value of the best solution achieved by CPLEX (if CPLEX finishes before the time limit, it is the optimal solution), t^* shows the CPU time (seconds) needed by CPLEX to reach that solution when solving the LOP-C model and *Gap* shows the difference between the integer and the linear relaxation problem. From the values in columns, *Gap* follows that LOP-C is more difficult than LOP. When $m = 4$, the problem has small linear gap, but when m increases, this linear gap also increases. When $m = 10$, the average linear gap is 36.59 which is larger than the common upper bound for the linear gap of LOP, i.e., 0.33 ($= 4/3 - 1$). Anyway, the *Gap* when $m = 20$ and when $m = 50$ uses the best integer solution within the time limit.

The addition of valid inequalities in Remark 3.2 as well as the heuristic separation of the 3-fence inequality in Remark 3.4 have failed to reduce the linear gap or the CPU time.

Now, we check the goodness of the metaheuristic proposed and described in Sect. 4 (Algorithm 1). The interest is on the efficiency of the method, that is, the quality of the solutions in a given computation time.

First with the results of Table 4, we have measured the average computation time needed by CPLEX to solve the instances in each group, $t_{aver_m}^*$. Then, we have run MH_LOP-C three times for each instance, and the time limit imposed in each one of those executions depends on the problem size and it has been calculated by the following expression:

$$\min \left\{ t_{aver_m}^*, \log \left(\frac{n}{m} \right)^m \right\}.$$

Preliminary studies have indicated some appropriate values for the parameters of the metaheuristic algorithm, and those are the ones employed in all the executions of MH_LOP-C: $Pop_size = 100$, $RefSet_size = 10$, $\alpha = 0.5$, $P_{mut1} = 0.1$, and P_{mut2} depends on m , being $P_{mut2} = 0.05$ for instances with 4 clusters, $P_{mut2} = 0.025$ for the instances with $m = 10$, $P_{mut2} = 0.0125$ for instances where the number of clusters is 25 and $P_{mut2} = 0.00625$ for 50 cluster instances.

Tables 5, 6, 7, 8 report the results for $m = 4, 10, 20, 50$, respectively. Headings are as follows. The first column is the instance number, v^* shows the objective value of the best solution achieved by CPLEX (if CPLEX finishes before the time limit, it is the optimal solution), and t^* shows the CPU time (seconds) needed by CPLEX to reach that solution when solving the LOP-C model. The third block of columns, called BEST SOLUTION, shows the results of the best among the three executions performed by MH_LOP-C. This block is divided into four columns: v_{BS}^* shows the best solution achieved, t_{BS}^* shows the CPUs time (seconds) to find this solution, #

Table 4 $m = 4, 10, 20$ y 50 (time limit: CPLEX 3 h)

Instance	m = 4			m = 10			m = 20			m = 50		
	v^*	t^*	Gap	v^*	t^*	Gap	v^*	t^*	Gap	v^*	t^*	Gap
1	463	2.19	8.61	2253	391.21	37.57	6813	10,800.00	59.76	28,958	10,800.00	68.53
2	486	2.01	2.87	2243	370.91	36.79	6790	10,800.00	57.02	29,800	10,800.00	62.52
3	477	2.64	5.98	2238	419.62	38.96	6790	10,800.00	60.98	29,643	10,800.00	68.70
4	466	2.60	8.21	2271	405.44	38.69	6734	10,800.00	62.42	29,100	10,800.00	68.61
5	465	1.20	7.00	2262	55.10	36.22	6783	10,800.00	59.31	28,497	10,800.00	73.27
6	462	2.30	8.62	2223	381.90	37.63	6736	10,800.00	61.15	28,884	10,800.00	71.95
7	469	2.35	7.27	2237	409.46	37.78	6680	10,800.00	61.65	29,247	10,800.00	68.85
8	494	0.42	1.03	2266	384.29	35.85	6737	10,800.00	60.36	27,839	10,800.00	77.24
9	469	0.54	6.90	2277	339.18	37.04	6712	10,800.00	62.59	29,303	10,800.00	69.96
10	448	2.11	11.18	2280	398.43	34.85	6830	10,800.00	58.66	29,559	10,800.00	66.96
11	480	1.23	2.98	2275	46.37	34.45	6565	10,800.00	63.98	28,883	10,800.00	70.76
12	468	1.65	7.04	2278	350.45	36.03	7021	10,800.00	53.32	29,146	10,800.00	67.67
13	466	2.29	8.63	2246	366.01	37.42	6755	10,800.00	60.41	29,209	10,800.00	67.57
14	456	2.15	8.11	2190	391.32	39.04	6599	10,800.00	61.12	29,842	10,800.00	62.78
15	452	2.73	11.33	2204	426.02	40.80	6750	10,800.00	62.46	28,348	10,800.00	75.73
16	466	1.78	5.54	2162	441.76	40.64	6439	10,800.00	66.62	29,533	10,800.00	65.76
17	463	2.07	10.48	2280	36.91	34.63	6471	10,800.00	65.67	28,260	10,800.00	72.02
18	480	1.96	5.85	2292	70.32	34.81	6790	10,800.00	59.55	28,343	10,800.00	73.84
19	466	3.16	6.89	2332	369.05	32.53	6944	10,800.00	57.18	27,993	10,800.00	77.77
20	470	1.29	7.22	2332	388.85	33.66	6817	10,800.00	60.71	27,348	10,800.00	82.54
21	520	0.25	0.00	2229	377.29	38.33	6499	10,800.00	66.95	28,648	10,800.00	70.83
22	470	2.00	7.90	2315	386.10	34.18	6857	10,800.00	60.43	29,783	10,800.00	66.41
23	465	2.00	6.27	2239	377.21	35.18	6537	10,800.00	62.57	28,530	10,800.00	67.62

Table 4 (continued)

Instance	m = 4			m = 10			m = 20			m = 50		
	v^*	t^*	Gap	v^*	t^*	Gap	v^*	t^*	Gap	v^*	t^*	Gap
24	472	1.80	6.54	2256	378.08	37.49	6586	10,800.00	63.94	29,255	10,800.00	67.33
25	461	2.47	7.13	2252	51.03	34.23	6686	10,800.00	58.73	28,912	10,800.00	67.91
Average			6.78			36.59			61.10			70.13

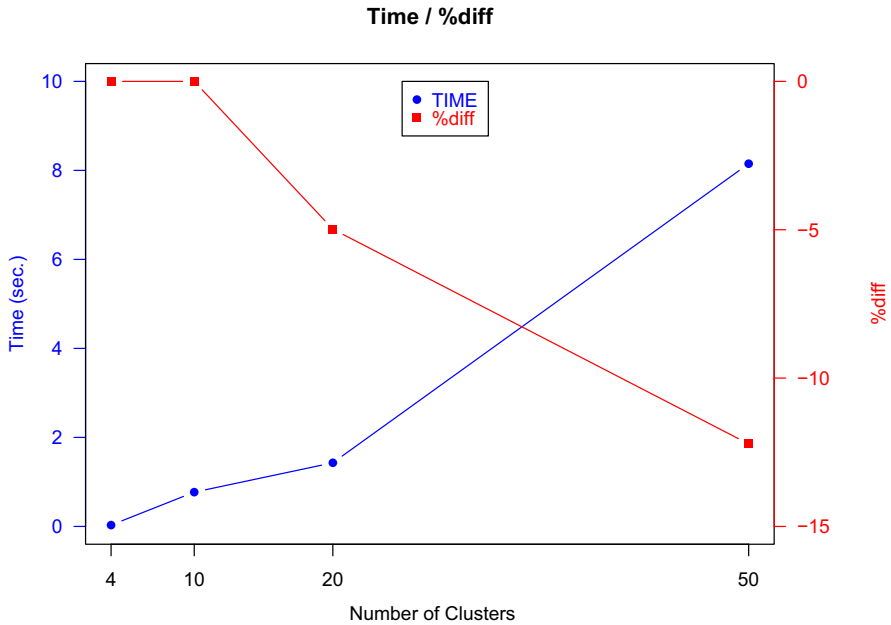


Fig. 2 Comparison of Tables 5, 6, 7, and 8

iter shows the number of iterations performed by the algorithm, and %diff shows the deviation from the solution reported by CPLEX, namely:

$$\% \text{ diff} = 100 \frac{v_{BS}^* - v^*}{v^*}. \tag{8}$$

Column #OPT in Tables 5 and 6 indicates the number of times among the three executions that MH_LOP-C finds the optimum and column #BST in Tables 7 and 8 reports the number of times that the metaheuristic finds, in the time limit considered, a solution which is better than the given by CPLEX. The last block of columns, *Average*, gives the averages for the three executions: average optimal value (v_A), average time (t_A), average number of iterations, and average deviation from the solution reported by CPLEX.

Table 5 reports the results for the case $m = 4$, in which all the clusters have 25 elements. In this case, the time limit imposed to the metaheuristic is, $t_{\text{aver}_4}^*$, the average time employed by CPLEX to solve the instances with 4 clusters. The results show that both methods obtain the optimal solution in the 25 instances of this group. The average time employed by CPLEX is 2 s and MH_LOP-C finds the optimum, in the best of the three executions, in 0.03 s, on average. Moreover, in all the instances, the metaheuristic proposed finds the optimal solution in less than one-fourth of a second, on average. Moreover, column #OPT, which reports the number of executions where the optimal solution is achieved by MH_LOP-C, shows that the metaheuristic reaches the optimum in all the three executions performed, which demonstrates its robustness and stability. The average number of iterations

Table 5 $m = 4$ (time limit: CPLEX 3 h, MH_LOP-C 2 s)

Instance	CPLEX sol.		Best solution				#OPT	Average			
	v^*	t^*	v_{BS}^*	t_{BS}^*	#iter	% diff		v_A	t_A	#iter	% diff
1	463	2.19	463	0.00	78,467	0.00	3	463	0.03	81,384	0.00
2	486	2.01	486	0.05	86,340	0.00	3	486	0.09	85,461	0.00
3	477	2.64	477	0.00	89,832	0.00	3	477	0.03	89,265	0.00
4	466	2.60	466	0.00	90,629	0.00	3	466	0.03	92,431	0.00
5	465	1.20	465	0.02	88,232	0.00	3	465	0.15	87,253	0.00
6	462	2.30	462	0.10	92,795	0.00	3	462	0.13	86,849	0.00
7	469	2.35	469	0.05	94,988	0.00	3	469	0.14	93,803	0.00
8	494	0.42	494	0.01	90,413	0.00	3	494	0.04	92,612	0.00
9	469	0.54	469	0.01	93,224	0.00	3	469	0.03	92,938	0.00
10	448	2.11	448	0.00	88,901	0.00	3	448	0.06	91,893	0.00
11	480	1.23	480	0.06	90,653	0.00	3	480	0.10	91,266	0.00
12	468	1.65	468	0.02	94,057	0.00	3	468	0.03	93,886	0.00
13	466	2.29	466	0.08	90,575	0.00	3	466	0.09	90,173	0.00
14	456	2.15	456	0.03	94,829	0.00	3	456	0.06	91,328	0.00
15	452	2.73	452	0.02	94,627	0.00	3	452	0.03	94,116	0.00
16	466	1.78	466	0.11	95,773	0.00	3	466	0.21	94,119	0.00
17	463	2.07	463	0.02	93,375	0.00	3	463	0.12	91,349	0.00
18	480	1.96	480	0.05	89,228	0.00	3	480	0.08	90,895	0.00
19	466	3.16	466	0.05	95,397	0.00	3	466	0.12	93,929	0.00
20	470	1.29	470	0.00	96,202	0.00	3	470	0.01	94,745	0.00
21	520	0.25	520	0.01	92,451	0.00	3	520	0.01	90,972	0.00
22	470	2.00	470	0.02	95,610	0.00	3	470	0.09	93,888	0.00
23	465	2.00	465	0.05	96,410	0.00	3	465	0.19	94,029	0.00
24	472	1.80	472	0.02	94,284	0.00	3	472	0.04	94,436	0.00
25	461	2.47	461	0.03	94,451	0.00	3	461	0.12	94,908	0.00
Average		2.00		0.03	92,069	0.00			0.08	91,517	0.00

carried out by MH_LOP-C is around 90,000, which means that it performs more than 1,000,000 iterations by second.

Table 6 shows the results for the case $m = 10$, that is, the instances have ten clusters with ten nodes per cluster in each instance. The average computation time for solving LOP-C with CPLEX is 320.12 s. Therefore, the time limit imposed to the metaheuristic is 10 s, $(\log(n/m))^m$, per execution. However, MH_LOP-C employs only 2.31 s on average for the instances in this group and, if we consider only the best of the three executions, less than 1 s on average. MH_LOP-C finds the optimal solution in the 25 instances and in 71 of the 75 executions performed. Only in 3 of the 25 instances, it does not achieve the optimal solution in all 3 executions. The average deviation of the solutions found by MH_LOP-C with respect to the optimal solution given by CPLEX is always smaller than 0.53%, employing a cpu time 140 times smaller. It varies from -1.9 to -6.53% in an average computation time

Table 6 $m = 10$ (time limit: CPLEX 3 h, MH_LOP-C 10 s)

Instance	CPLEX sol.		Best solution				#OPT	Average			
	v^*	t^*	v_{BS}^*	t_{BS}^*	#iter	% diff		v_A	t_A	#iter	% diff
1	2253	391.21	2253	2.96	83,354	0.00	3	2253.00	4.13	83,105	0.00
2	2243	370.91	2243	0.13	80,224	0.00	3	2243.00	0.50	81,703	0.00
3	2238	419.62	2238	1.31	80,794	0.00	3	2238.00	2.24	82,079	0.00
4	2271	405.44	2271	0.19	82,900	0.00	3	2271.00	4.40	84,062	0.00
5	2262	55.10	2262	0.79	82,726	0.00	3	2262.00	3.85	82,009	0.00
6	2223	381.90	2223	0.58	79,467	0.00	3	2223.00	2.01	80,471	0.00
7	2237	409.46	2237	1.64	76,937	0.00	1	2228.33	2.12	78,832	0.39
8	2266	384.29	2266	0.11	82,763	0.00	3	2266.00	0.27	77,820	0.00
9	2280	398.43	2280	5.20	79,392	0.00	3	2280.00	6.49	80,279	0.00
10	2277	339.18	2277	0.59	80,453	0.00	3	2277.00	3.11	79,316	0.00
11	2275	46.37	2275	0.30	80,162	0.00	3	2275.00	0.32	80,461	0.00
12	2278	350.45	2278	0.52	79,965	0.00	3	2278.00	1.28	80,233	0.00
13	2246	366.01	2246	0.61	80,590	0.00	3	2246.00	2.84	78,762	0.00
14	2190	391.32	2190	0.05	81,858	0.00	3	2190.00	2.86	80,857	0.00
15	2204	426.02	2204	0.63	80,104	0.00	2	2199.00	1.43	81,451	0.23
16	2162	441.76	2162	1.78	82,505	0.00	3	2162.00	4.13	81,982	0.00
17	2280	36.91	2280	0.14	80,928	0.00	3	2280.00	2.88	81,531	0.00
18	2292	70.32	2292	1.25	80,959	0.00	3	2292.00	5.73	82,310	0.00
19	2332	369.05	2332	0.20	83,062	0.00	3	2332.00	1.65	83,881	0.00
20	2332	388.85	2332	0.19	83,980	0.00	2	2319.67	0.18	84,002	0.53
21	2229	377.29	2229	0.22	76,985	0.00	3	2229.00	0.83	78,153	0.00
22	2315	386.10	2315	0.28	82,719	0.00	3	2315.00	2.22	83,413	0.00
23	2239	377.21	2239	0.30	82,033	0.00	3	2239.00	0.87	83,179	0.00
24	2256	378.08	2256	0.36	81,757	0.00	3	2256.00	1.10	83,848	0.00
25	2252	51.03	2252	0.05	83,265	0.00	3	2252.00	0.27	84,260	0.00
Average		320.12		0.77	81,195	0.00			2.31	81,520	0.05

which is 2245 times smaller. In this case the algorithm performs, on average, around 80,000 iterations to find the optimum, more than 35,000 iterations per second.

Table 7 summarizes the results for the instances with $m = 20$. In this case, CPLEX cannot find the optimal solution in any of the 25 instances in 3 h and column v^* reports the objective value of the best solutions achieved in that time. The time limit for MH_LOP-C is then 14 s. The metaheuristic needs, on average, 4.81 s to reach the best solution and only 1.43 s on average if only the best execution is considered. Negative values in column %diff indicate that the solution given by the metaheuristic is better than that the given by CPLEX. Moreover, column #BST indicates that MH_LOP-C always finds a solution which is better than the reported by CPLEX, in all the instances and all the executions performed per instance. Although in all cases, MH_LOP-C finds the best solution in less than 4 s, some time averages require nearly 14 s. For instance, the time needed by this algorithm to reach the best

Table 7 $m = 20$ (time limit: CPLEX 3 h, MH_LOP-C 14 s)

Instance	CPLEX sol.		Best solution				#BST	Average			
	v^*	t^*	v_{BS}^*	t_{BS}^*	#iter	% diff		v_A	t_A	#iter	% diff
1	6813	10,800.00	6999	3.43	46,174	-2.73	3	6986.33	5.55	46,951	-2.54
2	6790	10,800.00	7131	0.98	47,804	-5.02	3	7131.00	5.32	47,911	-5.02
3	6790	10,800.00	7082	0.84	46,269	-4.30	3	7071.33	2.42	46,273	-4.14
4	6734	10,800.00	6948	1.06	48,178	-3.18	3	6885.67	5.47	47,594	-2.25
5	6783	10,800.00	7118	3.04	47,582	-4.94	3	7088.67	3.51	47,443	-4.51
6	6736	10,800.00	7003	1.07	47,495	-3.96	3	6877.67	3.84	46,645	-2.10
7	6680	10,800.00	7041	0.23	48,516	-5.40	3	7020.33	0.89	45,225	-5.09
8	6737	10,800.00	6991	2.00	48,204	-3.77	3	6943.33	7.14	46,639	-3.06
9	6712	10,800.00	7137	1.38	42,904	-6.33	3	7073.00	5.81	44,135	-5.38
10	6830	10,800.00	7040	1.12	46,026	-3.07	3	6998.00	2.18	45,606	-2.46
11	6565	10,800.00	6873	2.10	47,733	-4.69	3	6846.67	8.65	47,252	-4.29
12	7021	10,800.00	7215	2.73	44,317	-2.76	3	7154.33	5.54	47,031	-1.90
13	6755	10,800.00	7187	0.32	47,953	-6.40	3	7055.67	1.77	47,975	-4.45
14	6599	10,800.00	6944	0.47	48,003	-5.21	3	6836.67	4.99	47,573	-3.60
15	6750	10,800.00	6977	2.47	42,920	-3.36	3	6971.67	6.28	44,353	-3.28
16	6439	10,800.00	6934	0.51	41,458	-7.69	3	6859.67	1.90	44,692	-6.53
17	6471	10,800.00	6931	0.32	48,694	-7.11	3	6886.33	2.67	47,290	-6.42
18	6790	10,800.00	7073	1.70	47,937	-4.17	3	7048.67	4.00	47,575	-3.81
19	6944	10,800.00	7130	3.85	45,863	-2.68	3	7121.67	9.26	46,515	-2.56
20	6817	10,800.00	7253	0.15	46,776	-6.40	3	7204.67	4.34	47,221	-5.69
21	6499	10,800.00	7003	2.53	42,357	-7.76	3	6912.00	4.97	43,168	-6.35
22	6857	10,800.00	7132	1.13	34,099	-4.01	3	7080.67	7.00	40,781	-3.26
23	6537	10,800.00	6873	1.10	45,846	-5.14	3	6777.33	6.82	44,735	-3.68
24	6586	10,800.00	7131	0.59	47,927	-8.28	3	6965.67	6.25	41,980	-5.76
25	6686	10,800.00	7106	0.62	46,469	-6.28	3	6873.00	3.74	44,281	-2.80
Average		10,800.00		1.43	46,039	-4.98			4.81	45,874	-4.04

solution for instance 22 is of 1.13, 13.98, and 5.89 s, respectively, that leads to an average computation time of 7 s. In this group of instances the ratio iterations by second is of $45,874/4.81 = 9537$.

Finally, Table 8 summarizes the results for the instances with $m = 50$ clusters and all the clusters of size 2. As CPLEX cannot find the optimal solution and it employs the 3 h in the solution process, the time limit for MH_LOP-C is then 15 s. The best option of each trio of iterations is always obtained in less than 14 s and in around 8 s on average. As in the previous table, values in column %diff are negative, and column #BST shows that MH_LOP-C finds better solutions than CPLEX in all the instances and all the executions. In the best execution for instance 20, MH_LOP-C outperforms CPLEX by about 23.73%. In this group of instances, the average differences vary between -5.68 and -23.09% in only 10.66 s, in contrast to the 10,800 s

Table 8 $m = 50$ (time limit: CPLEX 3 h, MH_LOP-C 15 s)

Instance	CPLEX sol.		Best solution				#BST		Average		#iter	% diff
	v^*	t^*	v_{BS}^*	t_{BS}^*	#iter	% diff	v_A	t_A				
1	28,958	10,800.00	31,884	3.88	8593	- 10.10	3	31,744.00	9.07	8672	- 9.62	
2	29,800	10,800.00	31,932	8.41	8419	- 7.39	3	31,944.00	9.77	8793	- 7.19	
3	29,643	10,800.00	32,781	9.23	9860	- 11.35	3	32,780.33	11.53	9300	- 10.58	
4	29,100	10,800.00	32,343	8.80	8816	- 12.14	3	32,471.00	9.12	9344	- 11.58	
5	28,497	10,800.00	32,417	7.16	9966	- 13.76	3	32,166.00	11.58	9835	- 12.88	
6	28,884	10,800.00	31,606	10.75	9471	- 11.28	3	31,823.33	13.38	9533	- 10.18	
7	29,247	10,800.00	32,235	9.75	9067	- 10.24	3	32,114.00	10.33	9531	- 9.80	
8	27,839	10,800.00	31,831	8.09	9449	- 17.05	3	32,299.67	11.08	9469	- 16.02	
9	29,303	10,800.00	33,005	3.64	9539	- 12.63	3	32,720.33	8.70	9534	- 11.66	
10	29,559	10,800.00	32,837	10.00	10,238	- 11.16	3	32,835.00	12.55	9955	- 11.08	
11	28,883	10,800.00	32,247	12.50	10,125	- 11.65	3	32,141.00	12.90	9917	- 11.28	
12	29,146	10,800.00	32,503	9.93	10,265	- 13.18	3	32,810.33	12.55	10,245	- 12.57	
13	29,209	10,800.00	32,530	6.83	9912	- 11.37	3	32,442.67	10.53	10,122	- 11.07	
14	29,842	10,800.00	31,437	8.49	9877	- 6.10	3	31,535.67	10.18	9967	- 5.68	
15	28,348	10,800.00	32,089	6.60	10,030	- 13.20	3	31,850.67	10.91	10,002	- 12.36	
16	29,533	10,800.00	31,866	10.61	9710	- 9.40	3	32,140.33	11.20	9743	- 8.83	
17	28,260	10,800.00	31,928	6.51	9816	- 12.98	3	31,772.67	9.31	9703	- 12.43	
18	28,343	10,800.00	32,364	5.85	10,194	- 16.83	3	32,604.33	9.83	10,205	- 15.03	
19	27,993	10,800.00	31,858	4.43	10,018	- 14.22	3	31,753.33	5.63	10,097	- 13.43	
20	27,348	10,800.00	33,677	4.68	9858	- 23.73	3	33,661.33	7.55	9997	- 23.09	
21	28,648	10,800.00	31,862	7.35	9499	- 11.22	3	31,790.67	11.56	9646	- 10.97	
22	29,783	10,800.00	32,559	12.07	9551	- 9.32	3	32,201.67	13.00	9819	- 8.12	
23	28,530	10,800.00	32,160	13.44	8817	- 12.72	3	31,980.00	14.01	8876	- 12.09	

Table 8 (continued)

Instance	CPLEX sol.		Best solution				#BST				Average	
	v^*	t^*	v_{BS}^*	t_{BS}^*	#iter	% diff	v_A	t_A	#iter	% diff		
2.4	29,255	10,800.00	32,626	8.39	9181	- 11.84	32,669.67	11.79	9499	- 11.67		
2.5	28,912	10,800.00	32,029	6.44	9149	- 10.78	31,859.67	8.57	9544	- 10.20		
Average		10,800.00		8.15	9577	- 12.22		10.66	9654	- 11.58		

employed by CPLEX. Now, the number of iterations per second has decreased until to 900.

Results in Tables 5, 6, and 7, 8 illustrate that the difficulty of the problem increases as m increases. With regards to CPLEX, somehow, it is due to the number of binary variables that the model has. If all the clusters are of size n/m , the LOP-C model has $n^2(m-1)/m + n$ binary variables. In particular, the number of binary variables is 7600, 9100, 9600, and 9900 for $m = 4, 10, 20,$ and 50 , respectively. In the case of the metaheuristic, the number of iterations per second decreases from 1 million to less than 1000, which implies that the problem is computationally hardest mainly as the number of clusters increases if the number of nodes remains unaltered. However, although t_A considerably increases with m , %diff decreases from zero to near -12%. Column %diff reports almost zero differences for the cases with 4 and 10 clusters, but these differences become negative values in the hardest instances, where the number of clusters is 20 and 50. Therefore, the difference between both algorithms is larger as the difficulty of the problem increases. Figure 2 depicts the evolution in the average time (t_A) and %diff for the different sizes considered in the computational study: computational time goes from 0 to 8 s, while the gap goes from values very close to zero to negatives values. For the easiest instances, with four or ten clusters, both resolution methods are effective, because the optimal solution is reached in all the instances, but MH_LOP-C is much more efficient, because this optimum is reached in a considerably much smaller computation time. When the difficulty of the problem increases, for instances with 20 or 50 clusters, CPLEX takes 3 h to reach solutions which are very far, in terms of quality, from the solutions reached by the metaheuristic in just a few seconds.

6 Conclusions

In this paper, we have presented the problem of linear ordering clusters of elements. The option of considering that only one representative of each cluster is required has been modeled through a binary linear formulation, LOP-C. A metaheuristic algorithm has been developed for approximating optimal solutions of LOP-C when the model itself requires too much computational time. The computational section illustrates the goodness of both the LOP-C model and the metaheuristic. The model allows the solution of problems with 9100 binary variables and the metaheuristic algorithm proposed reaches the optimal solution in the easy instances in most cases and outperforms, in seconds, the solution given by the exact optimizer within a time limit of 3 h in the hardest instances. This demonstrates the robustness and stability of the metaheuristic proposed. The computational section also gives some insights on the complexity that the number of clusters adds to the problem; when the same number of nodes are split in more clusters, the difficulty of the problem in terms of computational time for solving the problem increases.

Acknowledgements This work was supported by the Spanish Ministerio de Ciencia, Innovación y Universidades and Fondo Europeo de Desarrollo Regional (FEDER) through project PGC2018-099428-B-100

and by the Spanish Ministerio de Economía, Industria y Competitividad under Grant MTM2016-79765-P (AEI/FEDER, UE).

References

- Alcaraz J, Landete M, Monge JF (2012) Design and analysis of hybrid metaheuristics for the Reliability p-Median Problem. *Eur J Oper Res* 222:54–64
- Alcaraz J, Landete M, Monge JF, Sainz-Pardo JL (2019) Multi-objective evolutionary algorithms for a reliability location problem. *Eur J Oper Res*. <https://doi.org/10.1016/j.ejor.2019.10.043>
- Andoni A, Fagin R, Kumar R, Patrascu M, Sivakumar D (2008) Efficient similarity search and classification via rank aggregation. In: *Proceedings of the ACM SIG-MOD international conference on management of data*, pp 1375–1376
- Aparicio J, Landete M, Monge JF (2019) A linear ordering problem of sets. *Ann Oper Res*. <https://doi.org/10.1007/s10479-019-03473-y>
- Boussaid I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. *Inf Sci* 237:82–117
- 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
- Dwork C, Kumar R, Naor M, Sivakumar D (2001) Rank aggregation methods for the web. In: *Proceedings of the tenth international world wide web conference*, pp 613–622
- Fagin R, Kumar R, Sivakumar D (2003) Efficient similarity search and classification via rank aggregation. In: *Proceedings of the ACM SIGMOD international conference on management of data*, pp 301–312
- Fagin R, Kumar R, Mahdian M, Sivakumar D, Vee E (2004) Comparing and aggregating rankings with ties. In: *Proceedings of the ACM symposium on principles of database systems(PODS)*, pp 47–58
- Feng J, Fang Q, Ng W (2008) Discovering bucket orders from full rankings. In: *Proceedings of the 2008 ACM SIGMOD international conference on management of data*, pp 55–66
- Fortelius M, Gionis A, Jernvall J, Mannila H (2006) Spectral ordering and biochronology of european fossil mammals. *Paleobiology* 32:206–214
- García-Nové EM, Alcaraz J, Landete M, Monge JF, Puerto J (2017) Rank aggregation in cyclic sequences. *Optim Lett* 11:667–678
- Gionis A, Mannila H, Puolamaki K, Ukkonen A (2006) Algorithms for discovering bucket orders from data. In: *Proceedings of the ACM SIGKDD conference on knowledge discovery and data mining (KDD)*, pp 561–566
- Glover F (1977) Heuristics for integer programming using surrogate constraints. *Decis Sci* 8:156–166
- Glover F (1994) Genetic algorithms and scatter search unsuspected potentials. *Stat Comput* 4:131–140
- Glover F (1996) Tabu search and adaptive memory programming—advances, applications and challenges. In: Barr RS, Helgason RV, Dennington JL (eds) *Interfaces in computer science and operations research*. Kluwer Academic Publishers, Dordrecht, pp 1–75
- Glover F (1998) A template for scatter search and path relinking. In: Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D (eds) *Artificial evolution, Lectures Notes in Computer Science*, vol 1363. Springer, Berlin, pp 13–54
- Glover F, Laguna M (1997) *Tabu search*. Kluwer Academic Publishers, Dordrecht
- Glover F, Klastorin T, Kongman D (1974) Optimal weighted ancestry relationships. *Manag Sci* 20:1190–1193
- Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. *Control Cybern* 39:653–684
- Glover F, Laguna M, Martí R (2004) Scatter search and path relinking. *Foundations and advanced designs*. In: Onwubolu GC, Babu BV (eds) *New optimization techniques in engineering, Studies in Fuzziness and Soft Computing*, vol 141. Springer, Berlin, pp 87–100
- Grötschel M, Jäger M, Reinelt G (1984) A cutting plane algorithm for the linear ordering problem. *Oper Res* 32:1195–1220
- Halekoh U, Vach W (2004) A Bayesian approach to seriation problems in archeology. *Comput Stat Data Anal* 45:651–673
- Holland HJ (1975) *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor

- Kemeny J (1959) Mathematics without numbers. *Daedalus* 88:577–591
- Kendall M (1938) A new measure of rank correlation. *Biometrika* 30:81–89
- Laguna M, Martí R (2003) Scatter search. In: Sharda CR, Stefan Voss (eds) *Methodology and implementations*. Kluwer, Dordrecht
- Martí R, Reinelt G (2011) *The linear ordering problem exact and heuristic methods in combinatorial optimization*, 1st edn. Springer, Berlin
- Miklos I, Somodi I, Podani J (2005) Rearrangement of ecological matrices via markov chain monte carlo simulation. *Ecology* 86:3398–3410
- Mladenovic JA, Moreno-Perez JM, Moreno-Vega A (1996) A chain-interchange heuristic method. *Yugoslav J Oper Res* 6:41–54
- Puolamaki M, Fortelius M, Mannila H (2006) Seriation in paleontological data matrices via Markov chain Monte Carlo methods. *PLoS Comput Biol* 2:62–70
- Reinelt G (1985) *The linear ordering problem algorithms and applications*. Research and exposition in mathematics series. Heldermann, Berlin
- Teitz MB, Bart P (1969) Heuristic methods for estimating the generalized vertex median of a weighted graph. *Oper Res* 16:955–961
- Tromble R, Eisner J (2009) Learning linear ordering problems for better translation. In: *Proceedings of the 2009 conference on empirical methods in natural language processing*, vol 2, pp 1007–1016
- Yasutake S, Hatano K, Takimoto E, Takeda M (2012) Online rank aggregation. In: *JMLR workshop and conference proceedings*, vol 25. Asian conference on machine learning, pp 539–553

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.