Heuristics for the Minimum Linear Arrangement Problem

JUAN-JOSÉ PANTRIGO, ABRAHAM DUARTE
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
Juanjose.Pantrigo@urjc.es, Abraham.Duarte@urjc.es

VICENTE CAMPOS, RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
Vicente.Campos@uv.es, Rafael.Marti@uv.es

Version: November 21

ABSTRACT

The linear arrangement minimization problem consists of finding a labeling or arrangement of the vertices of a graph that minimizes the sum of the absolute values of the differences between the labels of adjacent vertices. This is a well-known NP-hard problem that presents a challenge to solution methods based on heuristic optimization. Many linear arrangement reduction algorithms have recently been developed and applied to structural engineering, VLSI and software testing. We undertake the development of different heuristic procedures with the goal of uncovering the most effective designs to tackle this difficult but important problem. Specifically, we consider the adaptation of constructive, local search, GRASP and Path Relinking methods for the linear arrangement minimization. We perform computational experiments with previously reported instances to first study the effects of changes in critical search parameters and then to compare the efficiency of our proposal with previous solution procedures.

Key Words: GRASP, Path Relinking, Metaheuristics

1. Introduction

Let $G=(V,E)$ be a graph with a vertex set V ($|V|=n$) and an edge set E ($|E|=m$). A labeling or linear layout f of G assigns the integers 1, 2, …, n to the vertices of G. Let $f(v)$ be the label of vertex v, where each vertex has a different label. The contribution of a vertex v, $L(v,f)$, to the objective function is the sum of the absolute values of the differences between $f(v)$ and the labels of its adjacent vertices. That is:

$$L(v,f)=\sum_{u\in N(v)}|f(v)-f(u)|$$

where $N(v)$ is the set of vertices adjacent to v. The linear arrangement of a graph G with respect to a labeling f is then:

$$LA(G,f)=\frac{1}{2}\sum_{v\in V}L(v,f)$$

The linear arrangement $LA(G)$ of graph G is thus the minimum $LA(G,f)$ value over all possible labelings f. In other words, the Linear Arrangement Minimization problem (MinLA) consists of finding a labeling f that minimizes $LA(G,f)$. This NP-hard problem (Garey and Johnson 1979) is related with two other well-known layout problems, the bandwidth and profile minimization problems. However, as pointed out by McAllister (1999), an optimal solution for one of these problems is not necessarily optimal for the other related problems. Therefore, in this paper we restrict our attention to methods and strategies specifically developed for the MinLA problem which was first stated by Harper (1964), though since then, many different algorithms have been proposed.

Juvan and Mohar (1992) introduced the Spectral Sequencing method (SSQ). This method computes the eigenvectors of the Laplacian matrix of G. It then orders the vertices according to the second smallest eigenvector. As stated by Petit (2003a), the rational behind the SSQ heuristic is that vertices connected with an edge will tend to be assigned numbers that are close to each other, thus providing a good solution to the MinLA problem.

McAllister (1999) proposed a heuristic method for the MinLA which basically consists of a constructive procedure that labels vertices in a sequential order. Vertices are selected according to their degree with respect to previously labeled vertices. This method compares favorably with previous methods for this and related problems.

Petit (2003a) reviewed lower bounds and heuristic methods, proposed new ones and introduced a set of 21 small and medium size instances ($62 \le n \le 10240$) for the MinLA. In particular, the author reviewed the Juvan-Mohar method (Juvan and Mohar 1992), the Gomory-Hu tree method (Adolphson and Hu 1973), the Edge method, and introduced the Degree method for improved lower bounds. Petit concluded that the Juvan-Mohar and Degree methods provide the best lower bounds; however, their values are far from the best known solutions, and therefore they are of very limited interest from a practical point of view. Regarding the heuristic methods, Petit considered both a constructive and a local search procedure. The Successive Augmentation (SAG) is a greedy heuristic that constructs step by step a solution extending a partial layout until all vertices have been enumerated. At each step, the best free label is assigned to the current vertex. Vertices are examined in the order given by a breadth-first search. Once a solution has been constructed, different improvement methods are considered. The author studied three different heuristics based on local search: Hill-climbing, Full-search and Simulated Annealing (SA). In the Hill-climbing method moves are selected at random; in the Full-search the entire neighborhood of a solution is examined, at each iteration, in search of the best available move. Finally, the SA algorithm implements the temperature parameter as described in Kirkpatrik et al. (1983) for move selection. Petit considered two neighborhoods, called flip2 and flip3. The former exchanges the label of two vertices, while the latter "rotates" the label of three vertices.

The experimentation in Petit (2003a) shows that the neighborhood based on the exchange of two labels (flip2) produces better results than the rotation (flip3). Overall experimentation concludes that the SA method outperforms the others, although it employs much longer running times (not reported in the paper). Therefore, the author recommends employing the Hill-climbing as well as the Spectral Sequencing methods. In Petit (2003b), a more elaborate Simulated Annealing algorithm is proposed. The author introduces a new neighborhood, flipN, based on the Normal distribution of the distances between

the labels of vertices. The Simulated Annealing algorithm based on the flipN neighborhood (SAN) improves upon the previous SA method. Moreover, to speed up the method, the initial solution is obtained with the SSQ algorithm. The combined method, SSQ+SAN, is able to outperform previous methods.

Rodríguez-Tello et al. (2007) proposed a new algorithm based on the Simulated Annealing methodology. The Two-Stage Simulated Annealing (TSSA) performs two steps. In the first one a solution is constructed with the procedure by McAllister (1999); then in the second step it performs a Simulated Annealing procedure based on exchanges of labels. This method introduces two new elements to solve the MinLA: a combined neighborhood and a new evaluation function. Given a vertex v, the first neighborhood, selected with a probability of 0.9, examines the vertices u such that their label $f(u)$ is close to the median of the vertices' labels adjacent to v (at a maximum distance of 2). The second neighborhood, selected with a probability 0.1, exchanges the labels of two vertices selected at random with diversification (exploration) purposes. The method evaluates the solutions with a function that is more discriminating than the original LA. The authors compared their TSSA method with the best known algorithms and they concluded that their method outperforms previous algorithms.

In this paper we explore the adaptation of the GRASP and Path Relinking methodologies to solve the MinLA problem. In particular, in Section 2 we study the constructive methods and in Section 3 the local search, both together forming our GRASP algorithm for this problem. Section 4 is devoted to describing a mechanism to selectively apply the improvement method within GRASP in order to save computational effort. Finally, Section 5 introduces a path relinking method as a GRASP post-processing for search intensification. The paper finishes with the computational experiments and the associated conclusions.

2. Constructive Methods

A straightforward scheme to construct a solution consists of performing n steps, labeling a vertex at each step with the lowest available label. The so-called Frontal Increase Minimization (FIM) starts by creating a list of unlabeled vertices U, which at the beginning consists of all the vertices in the graph (i.e. initially $U=V$). The first vertex v is randomly selected from all those vertices in U and labeled as 1. In subsequent construction steps, the candidate list CL consists of all the vertices in U that are adjacent to at least one labeled vertex. A vertex v is randomly selected from CL, labeled with the next available label and deleted from U. The method finishes after n steps, when all the vertices have received a label.

McAllister (1999) proposed the following refinement of the FIM construction. Let L be the set of labeled vertices and let $d(v)$ be the degree of vertex v, then $d_L(v)$ represents the number of adjacent labeled vertices to v and $d_U(v)$ represents the number of adjacent unlabeled vertices to v. This variant of the FIM construction is based on the computation of $sf(v)=d_U(v)-d_L(v)$ as a way to measure the attractiveness of vertex v for selection. Figure 1 shows a pseudo-code of this method, that we will refer to as C1, in which the vertex with the lowest sf-value is selected from CL and assigned the lowest available label in each construction step. The procedure specifies a tie-breaking mechanism in step 7 of Figure 1. When more than one vertex in CL has the minimum sf-value, the method selects the oldest one (the vertex with maximum number of iterations in CL). McAllister (1999) tested this method against previous constructive methods based on the FIM strategy, showing its superiority.

1. Initially $L = \emptyset$ and $U=V$.
2. Select a vertex u randomly from U.
3. Assign the label l=1 to u. $L = \{u\}$, $U = U - \{u\}$

WHILE ($U \neq \emptyset$)

4. $l = l+1$
5. Construct $CL = \{ v \in U / (w,v) \in E \ \forall w \in L \}$
6. Compute $sf(v) = d_U(v)-d_L(v)$ for all v in CL
7. Select the vertex u in CL with minimum $sf(u)$
8. Label u with the label l
9. $U = U - \{u\}$, $L = L \cup \{u\}$

ENDWHILE

Figure 1. Pseudo-code of the constructive method C1

We propose now a GRASP construction based on the computation of *sf(v)*. GRASP, Greedy Randomized Adaptive Search Procedure, is a multi-start or iterative process in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored until a local optimum is found after the application of the local search phase (Resende and Ribeiro, 2003). At each iteration of the construction phase, GRASP maintains a set of candidate elements CL that can be feasibly added to the partial solution under construction. All candidate elements are evaluated according to a greedy function (*sf(v)* in our case) in order to select the next element to be added to the construction. A restricted candidate list (RCL) is created with the best elements in CL. This is the greedy aspect of the method. The element to be added to the partial solution is randomly selected from those in the RCL. This is the probabilistic aspect of the heuristic. Once the selected element is added to the partial solution, the candidate list CL is updated and its elements evaluated. This is the adaptive aspect of the heuristic. Figure 2 shows the pseudo-code of this GRASP construction, which we will call C2.

1. Initially $L = \emptyset$ and $U=V$.
2. Select a vertex u randomly from U.
3. Assign the label l=1 to u. $L = \{u\}$, $U = U - \{u\}$

WHILE $(U \neq \emptyset)$

4. $l = l+1$
5. Construct CL = $\{ v \in U / (w,v) \in E \ \forall w \in L \}$
6. Compute $sf(v) = d_U(v)-d_L(v)$ for all v in CL
7. Construct RCL= $\{ v \in CL / sf(v) \leq th \}$
8. Select a vertex u randomly in RCL
9. Label u with the label l
10. $U = U - \{u\}$, $L = L \cup \{u\}$

ENDWHILE

Figure 2. Pseudo-code of the constructive method C2

In the GRASP construction above, the parameter *th* represents a threshold on the quality of the elements. Specifically, the elements in CL with an *sf*-value lower than *th* are admitted to become part of RCL. This search parameter is computed as a percentage α of the rank of *sf* in CL:

$$th = msf + \alpha\,(Msf\text{-}msf) \quad , \quad msf = \min_{v \in CL} sf(v) \quad , \quad Msf = \max_{v \in CL} sf(v)$$

Note that if α is equal to 0, then *th=msf*, and the GRASP construction is equivalent to the McAllister method. On the other hand, if α is equal to 1, then RCL=CL and the GRASP construction is equivalent to the FIM strategy. In the computational study reported in Section 6, we test the effect of changes in α to the solutions with this method.

In the adaptation above of the GRASP construction to the MinLA problem, we only consider the evaluation given by the *sf* function. We now propose a second variant, C3, in which we include the contribution of the selected vertex to the objective function. Specifically, let *C(v,l)* be the contribution of *v*, when *v* is labeled with label *l*, to the current solution (and the labels 1 to *l*-1 have already been assigned). In mathematical terms:

$$C(v,l) = \sum_{u \in N(v) \cap L} |f(u) - l|$$

The GRASP construction C3 only considers the vertices with minimum *sf*-value at each step. The restricted candidate list RCL is now formed with the vertices with minimum *sf*-value and with a *C*-value below a threshold th_C. In this way we can say that the *C*-value is used as a tie-breaking mechanism when more than one vertex in CL reaches the minimum *sf*-value. Figure 3 shows the pseudo-code of this construction.

1. Initially $L = \emptyset$ and $U=V$.
2. Select a vertex u randomly from U.
3. Assign the label l=1 to u. $L = \{u\}$, $U = U - \{u\}$

WHILE $(U \neq \emptyset)$

4. $l = l$+1
5. Construct CL = $\{ v \in U / (w,v) \in E \ \forall w \in L \}$
6. Compute $sf(v) = d_U(v)-d_L(v)$ for all v in CL
7. Construct CLmsf = $\{ v \in$ CL / $sf(v)=msf\}$
8. Construct RCL= $\{ v \in$ CLmsf / $C(v) \leq th_C \}$
9. Select a vertex u randomly in RCL
10. Label u with the label l
11. $U = U - \{u\}$, $L = L \cup \{u\}$

ENDWHILE

Figure 3. Pseudo-code of the constructive method C3

The parameter th_C in C3 is the threshold that establishes the vertices with a relatively low contribution to the current solution. It is computed as a percentage β of its range in the candidate list CLmsf:

$$th_C = dm_L + \beta (dM_L - dm_L) \ , \ dm_L = \min_{v \in CLmsf} C(v,l) \quad , \quad dM_L = \max_{v \in CLmsf} C(v,l)$$

If the parameter β in C3 takes the value 1, th_C equals dM_L and all the vertices in CLmsf are in RCL, thus resulting in a random selection among them. On the other hand, if it takes the value 0, th_C equals dm_L, thus resulting in a greedy selection. In Section 6 we will compare the three constructive methods C1, C2 and C3 described in this section and study the influence of their parameters on their performance.

3. Improvement Method

Exchanges are used as the primary mechanism to move from one solution to another in our implementation. We have considered an improvement method based on the ejection chain methodology. Given a labeling f and two vertices u and v with labels $f(u)$ and $f(v)$ respectively, we define $move(u,v)$ as the exchange of the labels $f(u)$ and $f(v)$. Let g be the resulting labeling when $move(u,v)$ is performed. We can then compute the value of g, $LA(G,g)$, from the value of f, $LA(G,f)$, as:

$$LA(G,g)=LA(G,f) - MoveValue(u,v)$$

where $MoveValue(u,v) = L(u,f)+L(v,f)-L(u,g)-L(v,g)$. Therefore, the larger the $MoveValue$, the better the move.

Previous heuristics for the MinLA (Rodríguez-Tello et al. 2007) are also based on this exchange move and given a vertex u, they consider the median of the vertices' labels adjacent to u as the best label for exchange. Although the median minimizes the sum of the absolute value differences, the following example (see Figure 4) illustrates that the exchange with the "median label" is not necessarily the best choice.

Consider the partial graph depicted in Figure 4 in which vertex u has a relatively large contribution value $LA(u,f)$=|23-2|+ |23-7|+|23-14|=46, and we consider exchanging its label, 23, with another one. According to the rule above, we would consider the median of the labels of its adjacent vertices, $med(u)$, as the best label for exchange. In this example, $med(u)$ is 7 and corresponds to vertex w. Let g be the labelling after the exchange ($g(w)$ = 23 and $g(u)$ = 7); we then obtain $LA(u,g)$ = 28, thus reducing the contribution of u to the objective function LA. However, if we assign label 8 to vertex u (instead of 7), the contribution of u will be |8-2|+|8-7|+|8-14|=13. Moreover, we note that 7 is the best label for u when its adjacent vertices keep their current label, so there is no point in assigning label 7 to vertex u, and at the same time, replacing the label of an adjacent vertex of u. This example shows that it is better to consider a label close to the "median-label" and not assigned to an adjacent vertex, than the "median-label" itself.

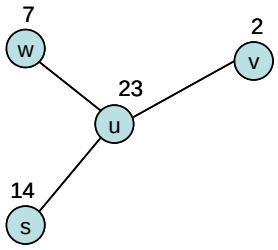

Figure 4. Move illustration

The situation illustrated in Figure 4 appears in all the cases in which the vertex considered for exchange has an odd degree (and thus the median value corresponds to the label of one of its adjacent vertex). This can be particularly problematic when the vertex u has only one adjacent vertex ($|N(u)|=1$). In that case, the selection of the median-label could even cycle the search. In line with this, previous papers (Rodríguez-Tello et al., 2007), do not limit the move to only considering *med(u)*, but examine a set of candidate labels close to the median at a maximum distance of 2. Therefore, they indirectly overcome this situation. We now propose to extend this set of "good labels" for exchanging including a search parameter *width* and avoiding the labels of adjacent vertices. The set *CL(u)* contains the candidate labels for u:

$$CL(u)=\{l \,/\, |l\text{-}med(u)| \leq width,\ l \neq f(v)\ \forall v \in N(u)\}.$$

Our improvement method is based on the ejection chain methodology. Glover and Laguna (1997) introduced the notion of *compound moves*, often called *variable depth methods*, constructed from a series of simpler components. Within the class of variable depth procedures, a special subclass called *ejection chain procedures* has recently proved useful. As described in Glover (1996), "Ejection chain procedures are based on the notion of generating compound sequences of moves, leading from one solution to another, by linked steps in which changes in selected elements cause other elements to be *ejected from* their current state, position or value assignment." In this section we adapt the notion of ejection chains to the MinLA problem.

In the MinLA problem, suppose that we want to exchange the label of a vertex u with the label *f(v)* of another vertex v because this exchange results in a reduction of *L(u,f)*. However, this exchange could produce an increase in *L(v,f),* thus resulting in a non-improving move (*MoveValue(u,v)*<0). We can therefore consider labeling u with *f(v)* but, instead of labeling v with *f(u)*, examine another vertex w and check whether its label *f(w)* is adequate for v (reducing *L(v,f)*). In terms of ejection chains, we may say that the assignment of *f(v)* to u caused *f(u)* to be "ejected" from u to w (assigning *f(w)* to v and defining a compound move of depth two that we can represent as *move(u,v)+move(v,w)*). Longer chains are possible by applying the same logic.

In our local search procedure based on ejection chains, EC, we define $move_{EC}(u,depth)$ as the ejection chain that starts searching for a good label for vertex u. To restrict the search and to make it more efficient, we only scan the labels in *CL(u)*. Let v be the vertex with an associated label $f(v) \in CL(u)$ with a maximum *MoveValue(u,v)*. The chain starts by making $move_{EC}(u,depth) = move(u,v)$, thus exchanging the labels *f(u)* and *f(v)*. If *move(u,v)* is an improving one, it is executed and the chain stops. Otherwise, we select the vertex v in *CL(u)* so that, when assigning its label *f(v)* to u, the contribution of u to the objective function, *L(u,f)* is minimized. We then search for a vertex w with a label *f(w)* in *CL(v)* which is adequate for v. We select the label *f(w)* in *CL(v)* with maximum *MoveValue(v,w)*. If the compound move of depth two, *move(u,v)+move(v,w),* is an improving one (*MoveValue(u,v)+MoveValue(v,w)*$\geq$*0*), the move is executed and the chain stops; otherwise the chain continues until the compound move becomes an improving one or the depth of the chain reaches the pre-specified limit *depth*. If none of the compound moves from depth 1 to *depth* examined in $move_{EC}(u,depth)$ is an improving move, no move is performed and the exploration continues with the next vertex to be considered for movement.

Note that we perform a move even when its value is 0. We have empirically found that this strategy permits the exploration of a larger number of solutions (compared to an implementation in which only moves with a positive value are performed), thus obtaining improved outcomes. On the other hand, we have also found that when we apply a move, we usually change a relatively large number of labels, and it

is useful to apply the ejection chain from the same vertex again. Therefore, we move to the next vertex considered for movement when no exchange is performed in the ejection chain.

A global iteration of the EC method consists of first ordering the vertices in the opposite order to their labeling in the construction. We then examine the vertices in this order in search of an improving $move_{EC}$ move. The rationale behind this ordering is to give priority to those vertices which are more constrained in the construction process. We have also considered a variant in which the vertices are ordered according to their contribution $L(u,f)$. However, this variant produces lower quality results compared with the former one. The method continues iterating only if in the previous iteration (i.e. the examination of all nodes) at least one improving move is performed. Otherwise, EC stops.

The ejection chain method EC has two parameters *width* and *depth* to control the distance between the labels and the number of vertices respectively involved in the move. Both parameters together permit the application of EC with a moderate running time; however, they constrain the search to "small" moves in the sense that it only considers candidate labels close to the neighbor's labels of the vertex selected for movement. In order to diversify the search and try more aggressive moves, we apply the Hill Climbing method (Petit 2003a) in which moves are selected at random at the end of EC as a post-processing of the ejection chain for *iter_Hc* iterations. Section 6 reports on the experimentation in which we study these search parameters.

4. GRASP Algorithm

As mentioned earlier, GRASP consists of a construction phase and an improvement phase. The most important element in the construction phase is that the selection at each step must be guided by a greedy function that adapts according to the selections at previous steps. In the computational experiments in Section 6 we compare the three constructive methods described in Section 2 and select the one employed in our GRASP algorithm. The improvement phase performs a sequence of moves, ejection chains in our EC method, towards a local optimum solution, which becomes the output of a complete GRASP iteration.

After a number of GRASP iterations, it is possible to estimate the percentage improvement achieved by the application of the improvement phase and use this information to increase the efficiency of the search (Laguna and Martí, 1999). Define the percentage improvement in the GRASP iteration i as:

$$P(i) = \frac{LA(G, f_i) - LA(G, f_i^*)}{LA(G, f_i)}$$

where f_i is the solution (labeling) constructed at iteration i, $LA(G,f_i)$ is its value, and f_i^* is the improved solution obtained applying the improvement method EC to f_i (and $LA(G,f_i^*)$ is its value). After n GRASP iterations, the mean μ_P and standard deviation σ_P of P can be estimated as:

$$\hat{\mu}_P = \frac{\sum_{i=1}^{n} P(i)}{n} \qquad \hat{\sigma}_P = \sqrt{\frac{\sum_{i=1}^{n} (P(i) - \hat{\mu}_P)^2}{n-1}}$$

Then, at a given iteration i, these estimates can be used to determine whether it is “likely” that the improvement phase will be able to improve the current construction enough to produce a better solution than the current best, f_{best}. In particular, we calculate the minimum percentage improvement $imp(i)$ that is necessary for a construction f_i to be better than f_{best}, as:

$$imp(i) = \frac{LA(G, f_i) - LA(G, f_{best})}{LA(G, f_i)}$$

If the value of $imp(i)$ is close to the estimation of μ_P, we can consider that when we apply the improvement method EC to the current solution f_i, we will probably obtain a solution f_i^* which is better than f_{best}. Therefore, in order to save computational time, we only apply EC to promising solutions f_i according to this estimation. In mathematical terms, if $imp(i) < \hat{\mu}_P + \delta\hat{\sigma}_P$, then we apply EC to f_i;

otherwise, we discard f_i. The value of δ is a search parameter representing a threshold on the number of standard deviations away from the estimated mean percentage improvement. In Section 6, we perform a set of preliminary experiments to test the effect of different δ values on solution quality and speed.

5. Path Relinking

Our implementation of path relinking has two phases. In the first one, a set of elite solutions ES is generated with the GRASP method (i.e. a small set of high quality solutions). Instead of retaining only the best overall solution when running GRASP, this phase stores a set of good solutions obtained with the method. We use the value of 10 for the cardinality of ES as recommended in Piñana et al. (2004) in the context of the bandwidth minimization problem, but here we extend the meaning of good introduced in that paper to reflect both quality and also diversity. In the second phase of our Path Relinking implementation, we apply the relinking process to each pair of solutions in the elite set ES.

The Elite Set plays a similar role to the Reference Set in Scatter Search (Laguna and Martí, 2003), since its elements are combined (linked) to produce new good solutions. Therefore, this set should contain good solutions scattered in the search space. The initial reference set (*RefSet*) in standard Scatter Search implementations is constructed by selecting the |*RefSet*|/2 best solutions from a population of solutions *P* and adding to *RefSet* the |*RefSet*|/2 most diverse solutions in *P* \ *RefSet* one-by-one. In our solving procedure we can consider the population of improved solutions $P=\{f_1^*, f_2^*,\ldots,f_{|P|}^*\}$ obtained with the application of the GRASP method. We then initialize the set of elite solutions ES with the best five solutions in *P*, and instead of the one-by-one selection of diverse solutions applied in Scatter Search, we propose solving the maximum diversity problem (MDP) in order to obtain the five most diverse solutions in *P* \ *ES*.

The MDP consists of finding, from a given set of elements and corresponding distances between elements, the most diverse subset of a given size. The diversity of the chosen subset is given by the sum of the distances between every pair of elements. Since the MDP is a computationally hard problem, we employ the GC_2 method (Duarte and Martí 2007) because it provides a good balance between solution quality and speed, attributes that are important in order to embed it as part of the overall solving procedure.

In order to define a distance function we need to consider that *reverse* labelings (permutations) define equivalent MinLA solutions. In mathematical terms, let $p=(p_1, p_2, \ldots, p_n)$ and $q=(q_1, q_2, .., q_n)$ be two labelings of a set of *n* vertices in which $q_i=n-p_i+1$, then we say that *p* and *q* are *reverse* permutations and equivalent MinLA solutions with the same value (it is easy to check that $LA(G,p)=LA(G,q)$). In order to measure the distance between pairs of MinLA solutions, we want equivalent solutions to have a distance of 0. We then propose the following function:

$$d(p,q)=\sum_{i=1}^{n}\partial_i \quad \text{where} \quad \partial_i=\begin{cases}1 \; if \;\; p_i \neq q_i \quad and \quad p_i \neq n-q_i+1\\ 0 \; otherwise\end{cases}$$

To illustrate this distance function, consider for example *p=(6,1,2,3,4,5)* and *q=(1,2,3,4,5,6)*, then *d(p,q)*=0+1+1+0+1+1=4. Therefore, we build the Elite Set, ES, with the five best solutions and the five most diverse solutions (according to the distance above) from the set of solutions *P* obtained with the application of the GRASP method.

We apply the relinking process to each pair of solutions in the ES. Given the pair (*f*,*g*), we consider the path from *f* to *g* (where *f* is the *initiating solution* and *g* the *guiding one*). In this path we basically assign each vertex label one by one in the guiding solution to the initiating solution. Given the pair (*f*,*g*), let *C*(*f*,*g*) be the candidate list of vertices to be examined in the relinking process from *f* to *g*. At each step in this process, a vertex *v* is chosen from *C*(*f*,*g*) and labeled in the *initiating solution* with its label *g(v)* in the *guiding solution*. To do this, in the *initiating solution* we look for the vertex *u* with label *g(v)* and perform *move(u,v),* then vertex *v* is removed from *C*(*f*,*g*). The candidate set *C*(*f*,*g*) is initialized with a randomly selected vertex. In subsequent iterations, each time a vertex is selected and removed from *C*(*f*,*g*), its adjacent vertices are included in this candidate set of vertices.

In the Path Relinking methodology, it is convenient to add a local search exploration from some of the solutions visited in order to produce improved outcomes (Laguna and Martí, 1999; Piñana et al., 2004). We have applied the local search method based on ejection chains EC to some of the solutions generated in the path. Note that two consecutive solutions after a relinking step only differ in the labeling of two vertices. Therefore, it does not seem efficient to apply the local search exploration at every step of the relinking process. We introduce the parameter *pr* to control the application of the EC method. In particular, EC is applied *pr* times in the relinking process. We report on the effectiveness of the procedure with different values of this parameter in the computational testing that follows. After the application of the Path Relinking strategy, we return the best solution found as the output of the method.

6. Computational Experiments

This section describes the computational experiments that we performed to compare our proposed procedure with state-of-the-art methods for solving the MinLA problem. We first describe our preliminary experimentation to compare the alternative solving methods proposed in previous sections. We also adjust the search parameters in order to establish the "best" combination of the proposed elements and strategies in our "final" algorithm. All experiments were performed on a personal computer with a 3.2 GHz Intel Xenon processor and 2.0 GB of RAM. We consider a set of instances previously introduced:

Petit Instances - This data set consists of twenty one instances ($62 \leq n \leq 10{,}240$ and $125 \leq m \leq 30{,}380$) introduced in Petit (2003a). They are designed to be difficult; i.e. in that they cannot be optimally solved by a brute force algorithm. This set includes random graphs and three families of "real-world" graphs: VLSI, graph-drawing and engineering (fluid-dynamics and structural mechanics).

We perform the preliminary experimentation on four instances: randomA1, c1y, bintree10 and mesh33x33. In our first preliminary experiment we compare the previous constructive method C1 (McAllister 1999) with our two methods C2(α) and C3(β) described in Section 2. We test five values for the search parameters α and β: 0.1, 0.2, 0.3, 0.4 and 0.5. We also test a variant C2(rand) in which the value of α is randomly selected from among these five values in each construction (and similarly with β in C3(rand)). Table 1 shows the results of these three methods on the four Petit instances mentioned above. We construct 100 solutions with each method on each instance and report the average across the four instances of the best solution found, Best, and the average of the worst solution found, Worst. We also report the average percentage deviation of the best solution obtained with each method from the best known solution, Dev. Best, as well as the average running time, Time.

Method	Best	Worst	Dev. Best	Time (seconds)
C1	53,9734.5	365,513.4	174.79%	2.25
C2(rand)	313,168.7	279,603.8	16.50%	2.75
C2(0.1)	313,168.7	279,516.6	16.50%	2.50
C2(0.2)	312,248.5	279,331.3	20.12%	2.75
C2(0.3)	312,932.7	279,401.6	19:97%	2.75
C2(0.4)	312,932.7	279,401.6	19:97%	2.75
C2(0.5)	313,168.2	279,344.8	16.50%	2.75
C3(rand)	327,343.5	282,040.0	13.77%	2.75
C3(0.1)	266,755.0	315,570.2	29.04%	2.50
C3(0.2)	266,852.7	322,400.5	31.71%	2.75
C3(0.3)	266,534.5	320,361.7	39.10%	2.75
C3(0.4)	265,768.2	321,176.7	36.76%	2.75
C3(0.5)	262,767.2	324,549.7	37.04%	2.75

Table 1. Comparison of constructive methods

The results in Table 1 indicate that the two proposed methods, C2 and C3, obtain better solutions than the previous method C1. Moreover, as shown in the average percentage deviation, the best results are obtained with the C3 method with a random selection of the parameter β (C3(rand)). However, in this experiment we have empirically found that some methods systematically obtain better solutions on some instances than other methods (although on average, across all the instances C3, is the best one). For

example, C1 always obtains the best solutions in the bintree instances and C2 always obtains the best in the random ones. Therefore we have considered a mixed method, called C4, in which we randomly select C1, C2(rand) or C3(rand) in each construction. C4 is able to obtain an average percentage deviation of 11.26% across the four problems of our preliminary experimentation, thus improving upon the other methods. We will then consider C4 as the constructive method in the rest of our experimentation.

In our second preliminary experiment we undertake to compare the effectiveness of the improvement method described in Section 3. As a baseline method, we consider the construction method C4 without any improvement run for 500 seconds. We compare it with the combination of C4 and the ejection-chain-based improvement method EC with several values for its two parameters: *width* and *depth*. To do this, we run C4+EC(*width,depth)* for 500 seconds on each instance and report, as in the previous experiment, the Best, Worst and Dev. Best average values.

Method	Best	Worst	Dev. Best
C4	263,636.7	509,565.2	11.26%
C4+EC(1,1)	264,574.5	457,632.0	13.34%
C4+EC(1,5)	263,224.8	327,512.8	13.15%
C4+EC(1,10)	262,739.3	370,197.0	13.19%
C4+EC(1,15)	264,512.5	319,452.0	13.58%
C4+EC(1,20)	262,872.3	316,349.0	13.05%
C4+EC(5,1)	262,025.0	454,472.5	11.45%
C4+EC(5,5)	261,836.8	309,190.0	11.21%
C4+EC(5,10)	263,258.3	303,643.8	11.23%
C4+EC(5,15)	260,298.3	324,966.8	11.14%
C4+EC(5,20)	262,929.3	332,535.3	11.30%
C4+EC(10,1)	261,698.0	302,020.5	10.51%
C4+EC(10,5)	260,345.3	313,030.5	10.37%
C4+EC(10,10)	263,572.3	439,862.0	10.88%
C4+EC(10,15)	262,983.8	311,830.8	10.84%
C4+EC (10,20)	261,869.8	452.744.8	10.91%

Table 2. Comparison of constructive and improvement methods

Table 2 shows that the best combination of the search parameters is obtained with *width*=10 and *depth*=5 in our ejection-chain procedure, since C4+EC(10,5) presents an average percentage deviation from the best known solution of 10.37. It should be noted that previous studies (Rodriguez-Tello et al. 2007) are limited to a *width* value of 5 in the local search method, and we are obtaining the best results with *width* set to 10. On the other hand, the variant C4+EC(1,1) is simply C4 plus a local search and surprisingly obtains worse solutions (13.34 average percentage deviation from best) than C4 by itself (which as shown in the previous experiment obtains an 11.26 average percentage deviation from best). This is explained by the fact that the local search method is extremely time-consuming in this problem, and within 500 seconds the method can only construct and improve less than 50 solutions in some instances.

As described in Section 3 the application of the improvement method finishes with a post-processing consisting of the Hill climbing method (HC). It is performed for *iter_Hc* iterations. Table 3 reports the statistics Best, Worst, Dev. Best as well as the CPU running time, Time, of the C4+EC+HC procedure with *iter_Hc*=0, n/20, n/15, n/10 and n/5.

iter_Hc	Best	Worst	Dev. Best	Time
0	261,538.5	434,911.3	10.12%	505.8
|V|/20	255,043.3	288,956.0	10.02%	502.0
|V|/15	254,594.5	298,544.3	9.64%	504.8
|V|/10	255,680.8	285,602.0	10.47%	508.5
|V|/5	253,935.5	284,989.3	9.96%	505.3

Table 3. Hill climbing post-processing in the improvement method

Table 3 clearly shows that the application of the HC post-processing in the EC improvement method presents a marginal improvement in the final quality of the solution. Specifically, the application of the Hill climbing method for n/15 iterations reduces the average percentage deviation from 10.12% to 9.64% consuming the same running time. In the reminder of our experimentation we call GRASP to the combination of the constructive method C4, the improvement method EC(10,5) and the post-processing HC.

In the third preliminary experiment we test the effect of the filter mechanism described in Section 4 in order to skip the improvement method within a GRASP iteration when the value of the construction recommends it. During the first 20 constructions the method stores the percentage improvement achieved with the local search procedure EC. Then, when a solution is constructed, it computes its minimum percentage improvement *imp* that is necessary to improve the best solution known so far. If $imp < \hat{\mu}_P + \delta\hat{\sigma}_P$ we apply the improvement method; otherwise it is skipped, where $\hat{\mu}_P$ and $\hat{\sigma}_P$ are the average and the standard deviation estimations of the percentage improvement. The parameter δ controls the number of standard deviations away from the estimated mean percentage improvement. To measure the effectiveness of this strategy and the best choice of δ, we run the GRASP algorithm for 100 iterations and apply this filter from iteration 21 to 100. Table 4 reports the average, across the instances in the preliminary experimentation, of the number of iterations in which the test recommends not applying the improvement, Skip, the average percentage deviation from the best known solution, Dev. Best and the average running time of the methods, Time.

δ	# Skip	Dev. Best	Time
0.5	51.7	10.51%	292.0
1	37.5	10.40%	550.2
1.5	30.0	10.90%	485.2
2	25.7	10.86%	485.7

Table 4. GRASP filter

Table 4 shows that as δ decreases the number of skipped improvement iterations increases. This is to be expected by definition of δ. However, the average percentage deviation from the best solution known, Dev. Best, does not present significant changes across different values of δ, thus indicating the robustness of the filter (i.e. solutions skipped for improvement hardly modify the final result). The best value for δ is 0.5, since it provides a saving in the CPU time without a significant deterioration of the quality of the final solution.

In our final preliminary experiment we test the path relinking (PR) algorithm. We use a size of 10 for the set of elite solutions ES as recommended in Piñana et al. (2004). We populate ES with the solutions obtained by applying our GRASP algorithm for 100 iterations. We then apply the PR to all the pairs in ES and report the best solution found. As described in Section 5, we apply the improvement method (EC(10,5)+HC) to some of the solutions in the paths. In particular, it is applied *pr* times in each path. Table 5 reports the statistics Best, Dev. Best and Time of the GRASP+PR algorithm with different values of the *pr* parameter.

pr	best	Dev. Best	Time
5	254,964.5	9,59%	824.5
10	253,773.5	8,51%	1,041.7
15	253,516.5	8,00%	1,268.0
20	252,476.5	8,04%	1,521.7

Table 5. Path relinking

Results in Table 5 shows that as *pr* increases, the GRASP+PR algorithm is able to marginally improve the quality of the final solution. However, as expected, running times also increase since, as we have already mentioned, the application of the improvement method is time-consuming. We will set *pr*=15 because it represents a good balance between solution quality and speed.

In the final experiment we compare our GRASP+PR algorithm with the best published methods: the McAllister constructive method C1 coupled with the Hill-climbing algorithm HC (Petit 2003a), C1+HC,

the simulated annealing SAN (Petit 2003) and the two stage simulated annealing TSSA (Rodriguez-Tello et al. 2007). Table 6 shows the results in the 21 Petit instances. We run these methods once on each instance (with the exception of C1+HC that, considering its simplicity, is executed 1000 times) with their parameters adjusted to run for about 1000 seconds of CPU time.

	GRASP+PR		**C4+HC**		**SAN**		**TSSA**	
	Value	Dev.	Value	Dev.	Value	Dev.	Value	Dev.
randomA1	914882	2.31	950394	6.28	894205	0.00	948868	6.11
randomA2	6572444	0.00	6708192	2.07	6596880	0.37	6625307	0.80
randomA3	14336736	0.00	14463797	0.89	14346700	0.07	14441751	0.73
randomA4	1779181	1.17	1824564	3.75	1758560	0.00	1816732	3.31
randomG4	179138	0.00	206123	15.06	299571	67.23	185912	3.78
bintree10	4267	0.00	13951	226.95	14247	233.89	4440	4.05
hc10	523776	0.00	538116	2.74	540512	3.20	523776	0.00
mesh33x33	32703	0.00	35509	8.58	38481	17.67	33464	2.33
3elt	431737	0.00	1369880	217.30	867560	100.95	509337	17.9
airfoil1	322611	0.00	867560	168.92	1369880	324.62	392989	21.8
whitaker3	1307540	0.00	4857190	271.48	4857190	271.48	1313857	0.48
c1y	65084	4.23	70896	13.54	73867	18.30	62441	0.00
c2y	82665	4.38	89029	12.41	89029	12.41	79199	0.00
c3y	136103	9.66	144902	16.75	163785	31.96	124117	0.00
c4y	125720	9.19	146651	27.36	146651	27.36	115144	0.00
c5y	109279	12.71	122652	26.51	123891	27.79	96952	0.00
gd95c	506	0.00	529	4.55	506	0.00	507	0.20
gd96a	114377	18.83	107945	12.15	111144	15.47	96253	0.00
gd96b	1421	0.35	1527	7.84	1483	4.73	1416	0.00
gd96c	519	0.00	531	2.31	519	0.00	523	0.77
gd96d	2414	0.84	2399	0.21	2421	1.13	2394	0.00
Avg. Time	**529.14**		**552.90**		**1274.14**		**870.57**	
Avg. Dev.	**3.03%**		**49.89%**		**55.17%**		**2.97%**	
#Best	**11**		**0**		**4**		**9**	

Table 6. Comparison of best methods

Table 6 is split into two parts. In the upper part we can find the results of each algorithm on each particular instance in the Petit set. Specifically, we report the value of the best solution found, Value, and the percentage deviation, Dev., between Value and the value of the best solution found with the four methods under consideration. In the lower part of Table 6 we report the average running time of each method across the 21 instances, Avg. Time, as well as the average percentage deviation, Avg. Dev., and the number of best solutions, #Best, that each method is able to obtain.

Table 6 shows that C4+HC and SAN are clearly inferior to the other two methods considered in this comparison, since they obtain an average percentage deviation from the best known solution of 49.89 and 55.17 respectively. GRASP+PR and TSSA obtain an average percentage deviation of 3.03 and 2.97 respectively and 11 and 9 best solutions found respectively out of the 21 instances in this set. It should be noted that our GRASP+PR algorithm presents a running time of 529.14 seconds on average, which compares favorably with the 870.57 seconds of TSSA. Moreover, if we run TSSA with its parameters adjusted as in Rodriguez-Tello et al. (2007), some instances would demand more than four hours of CPU time, although the average percentage deviation would be reduced to 0.05.

7. Conclusions

We have developed a heuristic procedure based on the GRASP methodology to provide high quality solutions to the linear arrangement minimization problem. The procedure is coupled with a Path Relinking post-processing for improved outcomes over a long term horizon.

Overall experiments with previously reported instances were performed to first study the contribution of the different elements in our procedure and then to compare it with previous methods. Our preliminary experiments do show the merit of the proposed mechanisms, such as the use of ejection chains within GRASP (Section 3), the filter of low quality constructions (Section 4) and the selection of diverse

solutions for Path Relinking (Section 5), that we hope other researchers might find effective in different combinatorial optimization problems. The resulting GRASP with Path Relinking implementation is highly effective, often surpassing the best procedures in the literature when solving the problem within moderate running times (1000 seconds on medium sized instances). On the other hand, if running time is not limited at all, the previous simulated annealing method by Rodriguez-Tello et al. (2007) obtains the best solutions on average.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (Grant Refs. TIN2005-08943-C02-02, TIN2006-02696), by the Comunidad de Madrid – Universidad Rey Juan Carlos project (Ref. URJC-CM-2006-CET-0603) and by the Generalitat Valenciana (Ref. GV/2007/047).

The authors thank Professors Rodriguez-Tello and Hao for sharing their results with them.

References

Adolphson, D. and T. C. Hu. (1973) "Optimal linear ordering", *SIAM Journal on Applied Mathematics*, 25(3):403–423.

Garey, M.R. and D.S. Johnson (1979) "*Computers and Intractability. A guide to the theory of completeness*", W. H. Freeman and Company, New York.

Glover, F. (1996) "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems", *Discrete Applied Mathematics* 65(1-3): 223-253.

Glover, F. and M. Laguna (1997) "*Tabu Search*", Kluwer Academic Publishers, Boston.

Harper, L. H. (1964) "Optimal assignment of numbers to vertices", *SIAM Journal on Applied Mathematics*, 12(1):131–135.

Juvan, M. and B. Mohar (1992) "Optimal linear labelings and eigenvalues of graphs", *Discrete Applied Mathematics*, 36(2):153–168.

Kirkpatrick, S., C. Gelatt, and M. Vecchi (1983) "Optimization by simulated annealing", *Science*, 220:671–680.

Laguna, M. and R. Martí (1999) "GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization", *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 44-52.

Laguna, M. and R. Martí (2003) "*Scatter Search: Methodology and Implementations in C*", Kluwer Academic Publishers, Boston.

Martí, R., M. Laguna, F. Glover and V. Campos (2001) "Reducing the Bandwidth of a Sparse Matrix with Tabu Search", *European Journal of Operational Research*, 135 (2), 211-220.

McAllister, A. J. (1999) "A new heuristic algorithm for the linear arrangement problem", Technical Report TR-99-126a, *Faculty of Computer Science, University of New Brunswick*.

Petit, J. (2003a) "Combining Spectral Sequencing and Parallel Simulated Annealing for the MinLA Problem", *Parallel Processing Letters* 13 (1), 71-91.

Petit, J. (2003b) "Experiments on the minimum linear arrangement problem", *ACM Journal of Experimental Algorithmics* 8.

Piñana, E., I. Plana, V. Campos and R. Martí (2004) "GRASP and Path Relinking for the Matrix Bandwidth Minimization", *European Journal of Operational Research* 153, 200-210.

Resende, M.G.C. and C.C. Ribeiro (2003) "Greedy Randomized Adaptive Search Procedures", in Handbook of Metaheuristic, Kluwer Academic Publishers, 219–249.

Rodriguez-Tello, E., J. Hao, J. Torres-Jimenez (2007) "An Effective Two-Stage Simulated Annealing Algorithm for the Minimum Linear Arrangement Problem", *Computers & Operations Research*, in press.