

Scatter Search and Path Relinking

A tutorial on the Linear Arrangement Problem

RAFAEL MARTÍ

Departamento de Estadística e Investigación Operativa, Universidad de Valencia,
Facultad de Matemáticas, Dr. Moliner 50, 46100, Burjassot, Valencia, España

Rafael.Marti@uv.es

Phone: (34) 96 354 30 90 / Fax: (34) 96 386 47 35

JUAN-JOSÉ PANTRIGO

Universidad Rey Juan Carlos. Esc. Tec. Sup. de Ing. Informática. Edif. Departamental II. C/ Tulipán
s/n. 28933 Móstoles, Madrid, España

juanjose.Pantrigo@urjc.es

Phone: (34) 91 488 73 93 / Fax: (34) 91 488 85 30

ABRAHAM DUARTE

Universidad Rey Juan Carlos. Esc. Tec. Sup. de Ing. Informática. Edif. Departamental II. C/ Tulipán
s/n. 28933 Móstoles, Madrid, España

Abraham.Duarte@urjc.es

Phone: (34) 91 488 81 16 / Fax: (34) 91 488 85 30

VICENTE CAMPOS

Departamento de Estadística e Investigación Operativa, Universidad de Valencia,
Facultad de Matemáticas, Dr. Moliner 50, 46100, Burjassot, Valencia, España

Vicente.Campos@uv.es

Phone: (34) 96 386 43 06 / Fax: (34) 96 386 47 35

FRED GLOVER

OptTek Systems, Inc, 2241 Seventeenth Street Boulder, CO 80302 USA,

Glover@OptTek.com

Phone: 303-447-3255 / Fax: 303-447-3886

ABSTRACT

Scatter search (SS) and path relinking (PR) are evolutionary methods that have been successfully applied to a wide range of hard optimization problems. The fundamental concepts and principles of the methods were first proposed in the 1970s and 1980s respectively, and were based on formulations, dating back to the 1960s, for combining decision rules and problem constraints. The methods use strategies for search diversification and intensification that have proved effective in a variety of optimization problems and that have sometimes been embedded in other evolutionary methods to yield improved performance. This paper examines the scatter search and path relinking methodologies from both conceptual and practical points of view, and identifies certain connections between their strategies and those adopted more recently by particle swarm optimization. We describe key elements of the SS & PR approaches and apply them to a hard combinatorial optimization problem: the minimum linear arrangement problem, which has been used in applications of structural engineering, VLSI and software testing.

KEY WORDS: Metaheuristics, scatter search, path relinking, particle swarm optimization, combinatorial optimization

1. Introduction

Scatter search (SS) was first introduced in Glover (1977) as a heuristic approach for general integer programming problems. SS systematically generates and updates a set of reference points that includes good solutions obtained by prior problem-solving efforts together with solutions that are screened to add diversity to the reference set. Path relinking (PR) was subsequently proposed in Glover (1989) and Glover and Laguna (1993) as an analog making use of neighborhood spaces in place of Euclidean spaces. Interesting relationships exist between the SS/PR approaches and the more recent particle swarm optimization methodology introduced by Kennedy and Eberhart (1995), as noted below.

The *SS and PR template* (Glover, 1997) has served as a foundation for most of the SS and PR implementations to date, underscoring processes to take advantage of the flexibility of the SS and PR methodologies. Through these processes, each of the basic components can be implemented in a variety of ways and degrees of sophistication, and hence can be adapted conveniently to a variety of different problem settings. Advanced options derive from the way that five pivotal elements of the methods are implemented. We can find a large number of papers on both the SS method and the PR method and their applications. Glover and Laguna (1997) provide overviews and a variety of references on these methods and the monographic book on Scatter Search (Laguna & Martí, 2003) together with the feature cluster of the European Journal of Operational Research (Martí, 2006), which includes 19 papers, provide the reader with the elements to design successful SS implementations. A recent survey of SS and PR methods appears in Resende et al (2010).

The following principles summarize the foundations of the scatter search and path relinking methodologies as evolved from its origins:

- Useful information about the form (or location) of optimal solutions is typically contained in a suitably diverse collection of elite solutions.
- When solutions are combined as a strategy for exploiting such information, it is important to provide mechanisms capable of constructing combinations that extrapolate beyond the regions spanned by the solutions considered.
- The manner of combining solutions may be viewed as forming paths between (and beyond) them (using Euclidean spaces in SS and neighborhood spaces in PR). Each path results in introducing attributes of one elite solution into another, by a process where the trajectory for a given solution is influenced by the guidance of other solutions.
- It is likewise important to incorporate heuristic processes to map combined solutions into new solutions. The purpose of these combination mechanisms is to incorporate both diversity and quality.
- Taking account of multiple solutions simultaneously, as a foundation for creating combinations, enhances the opportunity to exploit information contained in the union of elite solutions.

A connection between the SS & PR approaches and particle swarm optimization (PSO) arises through the SS & PR strategy of combining solutions in a manner that may be interpreted as generating trajectories of selected elite solutions by reference to directions determined by other elite solutions (called guiding solutions in path relinking). In contrast to PSO, the solutions guided by SS & PR are initially generated and subsequently updated after the combination process by the application of associated heuristic or metaheuristic processes. PSO methods characteristically guide the progress of solutions in different streams of search by reference to the personal best of each stream and the global best over all streams.

The evolving reference set of elite solutions used by SS & PR always includes the global best, and hence each solution in the set likewise enters into combinations with the global best to provide one of the influences on its trajectory, inviting a direct comparison with the PSO approach. However, each solution in SS & PR also enters into combinations with other members of the reference set, as would be analogous to a form of PSO guided by additional personal best solutions beyond those deriving from a single stream in question. Another apparent contrast is that SS & PR employ solution streams outside of those guided by other elite solutions, drawing on improving methods (such as iterated descent or tabu search) to yield new members to enter the elite reference set. Parallel versions of SS & PR which generate such streams simultaneously set up a master slave relationship where the maintenance of the reference set and the combinations produced from it yield the master processes than spawn candidate solutions passed to the

slaves employing local search. Aside from the differences in strategic detail, there are evidently intriguing similarities in overall conceptual design between SS & PR approaches and PSO methods when viewed from a higher level.

In this paper we consider the scatter search and path relinking methodologies applied to solve combinatorial optimization problems by targeting an NP-hard problem based on graphs; the minimum linear arrangement problem. We adapt three elements from the five in the SS and PR template that are context-dependent (i.e., that are conveniently susceptible to being customized for specific problem settings): the diversification generation method, the improvement method and the combination method. Accompanying this, we apply a standard design for the other two methods (which usually are context-independent): the reference set update and the subset generation methods. We begin by focusing on the SS method and later include elements of the PR method.

2. Problem Formulation

The minimum linear arrangement problem may be described as follows. 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 $1, 2, \dots, n$ 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 value of a linear arrangement of a graph G with respect to a labeling f is then:

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

Figure 1 shows an example of a linear arrangement and the associated computation of the objective function value. The optimum linear arrangement $LA(G)$ of graph G is then the minimum $LA(G,f)$ value over all possible labelings f – i.e., the Linear Arrangement Minimization problem (MinLA) consists of finding a labeling f that minimizes $LA(G,f)$. This NP-hard problem (Garey & Johnson, 1979) is related to 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.

The MinLA problem was first stated by Harper (1964), and through the years many different algorithms have been proposed for solving it, with varying degrees of success.

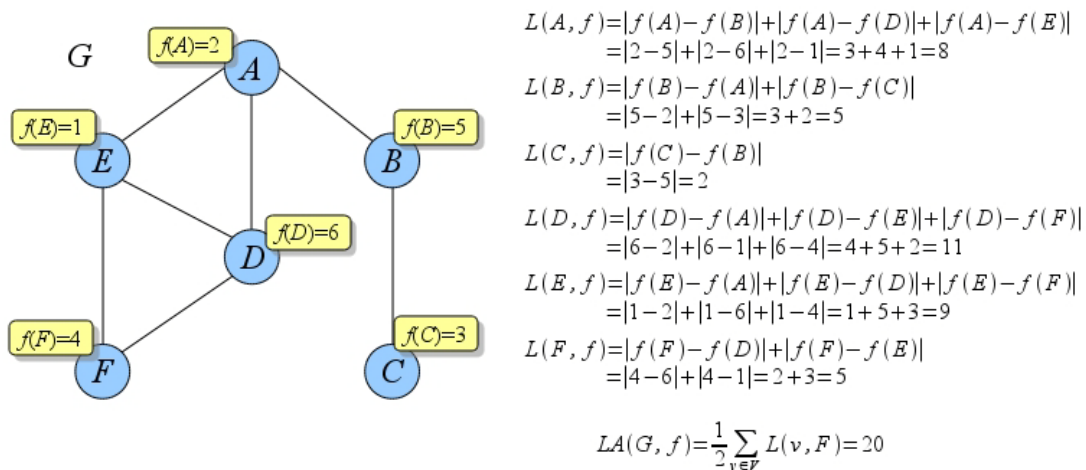


Figure 1: Objective function computation for a graph G and an arrangement f .

The remainder of this paper is organized as follows. In Section 3 we first give a brief summary of some of the leading methods previously developed from the MinLA problem. Section 4 covers the basics of the scatter search procedure, introducing the SS methodology and describing how we apply it to solve the MinLA problem. Section 5 describes the diversification generation methods we have incorporated which are based on the GRASP methodology and in Section 6 we describe an improvement method based on local search. Section 7 presents a mechanism to selectively apply the improvement method within Scatter Search in order to save computational effort. Finally, Section 8 introduces a path relinking procedure which fulfills the role of a combination method in our scatter search algorithm. The paper finishes with the computational experiments in Section 9 and associated conclusions in Section 10.

3. Brief Summary of Previous MinLA Methods

We chiefly restrict attention to more modern solution approaches proposed since 1990.

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 rationale 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 ($62 \leq n \leq 10240$) for the MinLA problem. Among the methods reviewed were the Juvan-Mohar method (Juvan & Mohar, 1992), the Gomory-Hu tree method (Adolphson and Hu 1973) and the Edge method to which he added a Degree method for improved lower bounds. Petit concluded that the Juvan-Mohar method and the Degree method provide the best lower bounds; however, their values are far from those of the best known solutions, and therefore they are of very limited interest from a practical point of view. Petit additionally introduced 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 Kirkpatrick, Gelatt, and Vecchi. (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.

Rodriguez-Tello, Hao and Torres-Jimenez (2008) 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.

4. Scatter Search Methodology

We begin by summarizing the “five-method template” for implementing SS and then discuss the relationships among its component processes. These key components are as follows:

1. A *diversification-generation method* to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.
2. An *improvement method* to transform a trial solution into one or more enhanced trial solutions. Neither the input nor the output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the “enhanced” solution is considered to be the same as the input solution.
3. A *reference-set update method* to build and maintain a *reference set* consisting of the b “best” solutions found (where the value of b is typically small, e.g. no more than 20), organized to provide efficient accessing by other parts of the method. Solutions gain membership to the reference set according to their quality or their diversity.
4. A *subset-generation method* to operate on the reference set, to produce several subsets of its solutions as a basis for creating combined solutions.
5. A *solution-combination method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

Figure 2 shows the interaction among these five components and highlights the central role of the reference set. This basic design starts with the creation of an initial set of solutions P , and then extracts the reference set (*RefSet*) of solutions from it. The shaded circles represent improved solutions resulting from the application of the improvement method.

The diversification-generation method is used to build a large set P of diverse solutions. The size of P ($PSize$) is typically at least 10 times the size of *RefSet*. The initial reference set is built according to the reference-set-update method, which takes the b “best” solutions from P to compose the *RefSet*, which typically is of modest size, e.g., containing between 10 and 30 solutions in all. The qualifier “best” is not restricted to referring to solution quality (i.e., the objective function value of a solution), but often embraces an expanded form of evaluation that accounts for diversity. This evaluation can change throughout the application of the procedure. For example, during a phase that is specifically devoted to diversification, the reference-set update method can consist of selecting b distinct and maximally diverse solutions from P . Regardless of the rules used to select the reference solutions, the solutions in *RefSet* are ordered according to quality, where the best solution is the first one in the list.

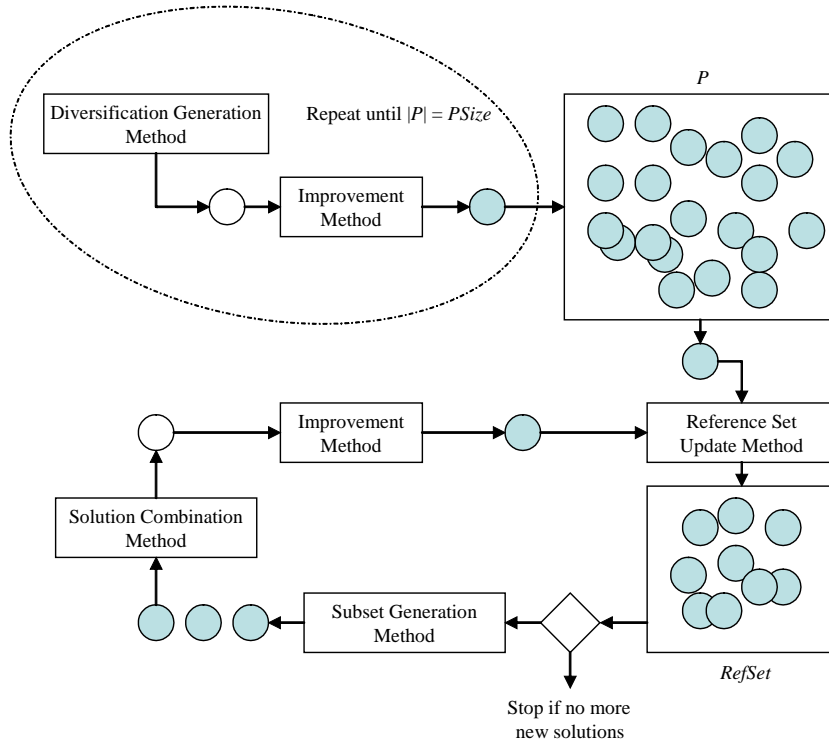


Figure 2: Schematic Representation of a Basic SS Design

The search is then initiated by applying the subset-generation method that, in its simplest form, involves generating all pairs of reference solutions. The pairs of solutions in $RefSet$ are selected one at a time and the solution-combination method is applied to generate one or more trial solutions which are subjected to the improvement method. The reference-set update method is applied once again to build the new $RefSet$ with the best solutions, according to the objective function value, from the current $RefSet$ and the set of trial solutions. The basic procedure terminates after all the subsets generated are subjected to the combination method and none of the improved trial solutions are admitted to $RefSet$ under the rules of the reference-set-update method.

In our approach, we use a simple mechanism to construct an initial reference set and then update it during the search. The size of the reference set is denoted by $|RefSet| = b_1 + b_2 = b$. The construction of the initial reference-set starts with the selection of the best b_1 solutions from P . These solutions are added to $RefSet$ and deleted from P . For each solution in $P \setminus RefSet$, the minimum of the distances to the solutions in $RefSet$ is computed. The solution with the maximum of these minimum distances is then selected. This solution is added to the $RefSet$ and deleted from P , and the minimum distances are updated. The process is repeated b_2 times, where $b_2 = b - b_1$. The resulting reference set has b_1 high-quality solutions and b_2 diverse solutions.

Of the five SS components, only four are strictly required. The improvement method is usually needed if high-quality outcomes are desired, but an SS procedure can be implemented without it. On the other hand, the SS method could incorporate tabu search or another complex metaheuristic as the improvement method (usually demanding more running time).

5. Diversification Generation Methods

A straightforward scheme to construct a solution consists of performing n steps, labeling a vertex at each step with the lowest available label. At a higher level of sophistication, 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 labeled vertices adjacent to v and $d_U(v)$ represents the number of unlabeled vertices adjacent 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 3 shows a pseudo-code of this method, denoted 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 3. 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  $u$ .  $L = \{u\}$ ,  $U = U \setminus \{u\}$ 
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.  $U = U \setminus \{u\}$ ,  $L = L \cup \{u\}$ 
ENDWHILE

```

Figure 3. 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 & 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 4 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  $u$ .  $L = \{u\}$ ,  $U = U \setminus \{u\}$ 
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.  $U = U \setminus \{u\}$ ,  $L = L \cup \{u\}$ 
ENDWHILE

```

Figure 4. 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 fraction α of the rank of sf in CL:

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

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 9, 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 5 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  $u$ .  $L = \{u\}$ ,  $U = U \setminus \{u\}$ 
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  $CL_{msf} = \{v \in CL / sf(v) = msf\}$ 
8. Construct  $RCL = \{v \in CL_{msf} / C(v) \leq th_C\}$ 
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 5. 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 fraction β of its range in the candidate list CL_{msf} :

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

If the parameter β in C3 takes the value 1, th_C equals dM_L and all the vertices in CL_{msf} 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 9 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.

6. 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 often used in association with tabu search (see Glover and Laguna, 1997). As a basis for describing our approach, it is useful to first discuss a type of move commonly used in heuristics for the MinLA problem.

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 MinLA based on this exchange move include the approach by Rodríguez-Tello et al. (2008), which, given a vertex u , considers the median of the vertices' labels adjacent to u as the best label for exchange. Although assigning the median label $med(u)$ to u minimizes the sum of the absolute values of differences between this label and the labels of adjacent vertices, it is of course necessary to give u a label different from $med(u)$ unless the adjacent vertex v with $f(v) = med(u)$ is given a new assignment different from $med(u)$. This will occur if vertex u and vertex v exchange labels, but then the rationale of reassigning u the label $med(u)$ is destroyed¹. The following example (see Figure 6) illustrates that an exchange with the “median label” is not necessarily the best choice.

Consider the partial graph depicted in Figure 6 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 v . Let g be the labelling after the exchange ($g(v) = 23$ 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 $|8-2| + |8-7| + |8-14| = 13$. This motivates an exchange between $f(u)$ and a label $f(v^*)$, where v^* is not adjacent to u and where $f(v^*)$ is close to the value $med(u)$. (In the present case, $f(u) = 23$ and $f(v^*) = 8$.) Of course, this is only a first order consideration, because it would be entirely possible that exchanging the indicated $f(u)$ and $f(v^*)$ could produce an unfavourable result – as where, for example, the vertices adjacent to v^* contain the labels 5, 6, 9 and 10. (Then assigning the label 23 to $f(v^*)$ would cause the new labeling of v^* to increase its LA value by 62, causing the value of the complete solution to deteriorate significantly.) Nevertheless, we abide by the principle that it is usually better to consider a label close to the “median-label” and not assigned to an adjacent vertex, than the “median-label” itself.

The situation illustrated in Figure 6 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., 2008), 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 :

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

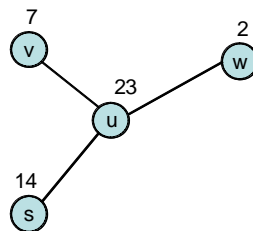


Figure 6. Move illustration

¹ In the case where a vertex has an even number of adjacent neighbors, so that the labels of two of these neighbors could be considered candidates for a median, then selecting any value between and including the two candidates will minimize the sum of the absolute values of the differences. This property is not restricted to label values that are positive integers, but holds for any finite collection of real valued labels

The ejection chain methodology we employ follows the general form indicated in Glover (1992) and Glover (1996), which is “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 make use of this notion as follows:

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 $LA(u,f)$. However, this exchange could produce an increase in $LA(v,f)$, thus resulting in a non-improving move ($MoveValue(u,v) < 0$). We can therefore consider labeling v with $f(u)$ but, instead of labeling u with $f(v)$, examine another vertex w and check whether the label $f(v)$ may be advantageously assigned to w and whether, to complete the process, the label $f(w)$ is appropriate to assign to u (as by reducing $LA(u,f)$). In terms of ejection chains, we may say that the assignment of $f(u)$ to v caused $f(v)$ to be “ejected” from v to w (and concluding by assigning $f(w)$ to u). The outcome defines a compound move of depth two that we can represent as $move(u,v)+move(v,w)$. We can then repeat this logic to build longer chains.

In our local search procedure based on ejection chains, EC, we define $move_{EC}(u,depth)$ as the ejection chain that starts the chain building process from 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, $LA(u,f)$ 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 length 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 $LA(u,f)$. 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.

In addition to the parameter $depth$ our ejection chain method EC includes the parameter $width$ described previously. By this means we control both the number of vertices involved in the move and the distance between the labels. 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 9 reports on the experimentation in which we study these search parameters.

7. Filtering Solutions

After a number of iterations, it is possible to estimate the fractional improvement achieved by the application of the improvement phase and use this information to increase the efficiency of the search (Laguna & Martí, 1999). Define the fractional improvement in the 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 iterations, the mean μ_p and standard deviation σ_p of P can be estimated as:

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

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 fractional 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 9, we perform a set of preliminary experiments to test the effect of different δ values on solution quality and speed.

8. Combination Method based on Path Relinking

We consider a reference set with $b = 10$ solutions and apply a combination method to them. In our implementation for the linear arrangement problem we have considered the path relinking methodology to elaborate our combination method.

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 $P \setminus RefSet$ one-by-one, where the concept of diversity is defined below. Instead of this one-by-one selection of diverse solutions we consider here solving the maximum diversity problem (MDP) in order to obtain the five most diverse solutions in $P \setminus RefSet$.

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 GC₂ method (Duarte & 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 $p = (p_1, p_2, \dots, p_n)$ and $q = (q_1, q_2, \dots, q_n)$ 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:

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

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 *RefSet* 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 *RefSet*. 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 & 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.

9. Computational Experiments

This section describes the computational experiments that we performed to compare our proposed procedure with previous 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 a good 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 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 an explicit enumeration of all their feasible solutions. This set includes random graphs and three families of "real-world" graphs: VLSI, graph-drawing and engineering (fluid-dynamics and structural mechanics).

Method	Best	Worst	Dev. Best	Time (seconds)
C1	53,9734.5	365,513.4	174.79%	2.25
C2(rand)	279,603.8	313,168.7	16.50%	2.75
C2(0.1)	279,516.6	313,168.7	16.50%	2.50
C2(0.2)	279,331.3	312,248.5	20.12%	2.75
C2(0.3)	279,401.6	312,932.7	19.97%	2.75
C2(0.4)	279,401.6	312,932.7	19.97%	2.75
C2(0.5)	279,344.8	313,168.2	16.50%	2.75
C3(rand)	282,040.0	327,343.5	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

We perform the preliminary experimentation on four instances: randomA1, c1y, bintree10 and mesh33x33 selected at random from the original set. 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 5. 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.

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.

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 C4 coupled with different improvement methods with a time limit of 500 sec.

In our second preliminary experiment we undertake to compare the effectiveness of the improvement method described in Section 5. 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. 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., 2008) 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 5 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, $|V|/20$, $|V|/15$, $|V|/10$ and $|V|/5$.

<i>iter_Hc</i>	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 remainder of our experimentation we select the constructive method C4 and the improvement method EC(10,5) with the post-processing HC.

In the third preliminary experiment we test the effect of the filter mechanism described in Section 7 to skip the improvement method when the value of the construction recommends it. During the first 20 constructions the method stores the fractional improvement achieved with the local search procedure EC. Then, when a solution is constructed, it computes its minimum fractional improvement imp that is necessary to improve the best solution known so far. If $im \geq \hat{\mu}_p + \delta \hat{\sigma}_p$ 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 populate the *RefSet* with 100 solutions 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.

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.

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. Filtering solutions

In our final preliminary experiment we test the path relinking (PR) algorithm to measure the contribution of the combination method within our scatter search procedure. We populate the *RefSet* with the solutions obtained by applying our GRASP algorithm for 100 iterations. We then apply the PR to all the pairs in *RefSet* and report the best solution found. As described in Section 8, 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 SS+PR algorithm with different values of the pr parameter.

Results in Table 5 show that as pr increases, the SS+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 set $pr=15$ because it represents a good balance between solution quality and speed.

<i>pr</i>	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. Scatter search with path relinking (SS+PR)

In the final experiment we compare our SS+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., 2008). 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.

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.

	SS+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 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. Our SS+PR method obtains the greatest number of best solutions found by any of the methods, 11, followed by the TSSA method which obtains 9 best solutions. SS+PR and TSSA obtain an average percentage deviation of 3.03 and 2.97 respectively. It should be noted that our SS+PR algorithm presents a running time of 529.14 seconds on average, which compares favorably with the 870.57 seconds of TSSA. Moreover, TSSA would demand more than four hours of CPU time in some instances if we adjusted its parameters as in Rodriguez-Tello et al. (2008), although the average percentage deviation from employing such lengthy runs would be reduced to 0.05.

We finish our experimentation studying the contribution of the different SS elements in the evolution of the best solution. Specifically, Figure 7 depicts, for each SS iteration, the value of the best solution in the

RefSet (Best RefSet), the value of the best solution obtained with the combination of the solutions in the *RefSet* (Best Comb), and the value of the best solution resulting from the application of the improvement method to the combined solutions (BestLS).

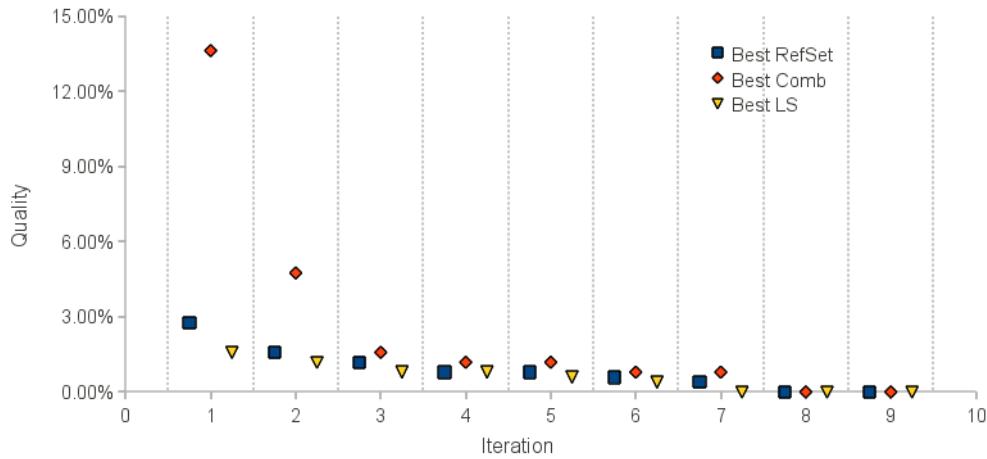


Figure 7. Evolution of the best SS solution

Figure 7 shows the contribution of the combination and improvement methods to the best solution found. This figure clearly shows that the solutions obtained by combination are not able to improve themselves the value of the best solution in the *RefSet*; however, they constitute good seeds for the application of the improvement method. This is especially true in iterations 1, 2 and 7 in which the application of the improvement method to the combined solutions is able to improve the best solution in the *RefSet*, thus generating a new best solution overall.

10. Conclusions

We have developed a heuristic procedure based on the SS methodology to illustrate how this methodology can be applied to an intensively studied problem in combinatorial optimization – the linear arrangement minimization problem. Although our development is tutorial in nature, our illustrative method nevertheless performs highly effectively compared to the leading methods in the literature.

Overall experiments with previously reported instances were performed to first identify the contribution of the different elements in our procedure and then to compare the outcomes with those of previous methods. Our preliminary experiments illustrate the merit of the proposed mechanisms, such as the use of ejection chains (Section 6), the filter of low quality constructions (Section 7) and the selection of diverse solutions for Path Relinking (Section 8), that we hope other researchers might find useful in different combinatorial optimization problems. The efficacy of the resulting scatter search with path relinking implementation is demonstrated by surpassing the best procedures in the literature in the number of best solutions obtained when solving the problem within moderate running times (1000 seconds on medium sized instances). An immediate opportunity for obtaining still better results may be noted to arise by employing a self-adjusting mechanism for choosing parameter values, rather than relying on fixed values that represent a compromise between competing considerations. To carry the basic foundations of this work further, a sequel will examine more finely tuned versions of the illustrated strategies, and additionally focus on longer term strategies, thus exposing options for obtaining improved solutions when time limits are made substantially longer.

Acknowledgments

This research has been partially supported by the *Ministerio de Ciencia e Innovación* of Spain (TIN2009-07516). The authors thank Professors Rodriguez-Tello, Hao and Torres-Jimenez for sharing their results with them.

References

- Adolphson, D., & Hu., T. C. (1973). Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3) 403-423.
- Garey, M.R., & Johnson, D.S. (1979). *Computers and Intractability. A guide to the theory of completeness*, New York: W. H. Freeman and Company.
- Glover, F. (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences* 8, 156-166.
- Glover, F. (1982). Ejection Chains, Reference Structures and Alternating Path Methods, Boulder: University of Colorado.
- Glover, F. (1989). "Tabu Search - Part I," *ORSA Journal on Computing*, 1(3) 190-206.
- 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. (1997) A Template for Scatter Search and Path Relinking. In Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., & Snyers, D., (Eds.), *Artificial Evolution, Lecture Notes in Computer Science* 1363, 13-54. Springer.
- Glover, F., & Laguna, M. (1993). Tabu Search. In *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, ed., Blackwell Scientific Publishing, pp. 71-140
- Glover, F., & Laguna, M. (1997). *Tabu Search*. Boston: Kluwer Academic Publishers.
- Harper, L. H. (1964). Optimal assignment of numbers to vertices. *SIAM Journal on Applied Mathematics*, 12(1) 131-135.
- Juvan, M., & Mohar, B. (1992). Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2) 153-168.
- Kennedy, J., Eberhart, R.C. (1995). Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. IV, pp. 1942-1948.
- Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220, 671-680.
- Laguna, M., & Martí, R. (1999). GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11(1) 44-52.
- Laguna, M., & Martí, R. (2003). *Scatter Search: Methodology and Implementations in C*. Boston: Kluwer Academic Publishers.
- Martí, R. (Guest editor) (2006). Scatter Search Methods for Optimization. A Feature Cluster of the *European Journal of Operational Research*, 169(2).
- Martí, R., Laguna, M., Glover, F., & Campos, V. (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. New Brunswick, CA: 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., Plana, I., Campos, V., & Martí, R. (2004) GRASP and Path Relinking for the Matrix Bandwidth Minimization, *European Journal of Operational Research* 153, 200-210.
- Resende, M.G.C. & Ribeiro, C.C. (2003). Greedy Randomized Adaptive Search Procedures. in *Handbook of Metaheuristic*, 219-249. Kluwer Academic Publishers.
- Resende, M.G.C., Ribeiro, C.C., Glover, F. & Martí, R (2010). Scatter search and path-relinking: Fundamentals, advances, and applications, *Handbook of Metaheuristics (2nd edition)*, M. Gendreau and J. Y. Potvin, eds. Kluwer Academic Publishers, 87-108, New York.
- Rodriguez-Tello, E., Hao, J., & Torres-Jimenez, J. (2008). An Effective Two-Stage Simulated Annealing Algorithm for the Minimum Linear Arrangement Problem. *Computers & Operations Research* 35(10) 3331-3346.