An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem

Eduardo Rodriguez-Tello\(^a,\)^*, Jin-Kao Hao\(^a\), Jose Torres-Jimenez\(^b\)

\(^a\)LERIA, Université d’Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
\(^b\)CINVESTAV-Tamaulipas, Information Technology Laboratory, Km. 6 Carretera Victoria-Monterrey, 87276 Victoria Tam., Mexico

Abstract

In this paper, an improved two-stage simulated annealing algorithm is presented for the minimum linear arrangement problem for graphs. This algorithm integrates several distinguished features including an efficient heuristic to generate good quality initial solutions, a highly discriminating evaluation function, a special neighborhood function and an effective cooling schedule. The algorithm is evaluated on a set of 30 well-known benchmark instances of the literature and compared with several state-of-the-art algorithms, showing improvements of 17 previous best results.

*Keywords:* Linear arrangement; Evaluation function; Heuristics; Simulated annealing

1. Introduction

The *minimum linear arrangement* (MinLA) problem was first stated by Harper in [1]. His aim was to design error-correcting codes with minimal average absolute errors on certain classes of graphs. Later, in the 1970s MinLA was used as an abstract model of the placement phase in VLSI layout, where vertices of the graph represented modules and edges represented interconnections. In this case, the cost of the arrangement measures the total wire length [2]. MinLA arises also in other research fields like biological applications, graph drawing, software diagram layout and job scheduling [3,4].

The MinLA problem can be stated formally as follows. Let \(G(V, E)\) be a finite undirected graph, where \(V (|V| = n)\) defines the set of vertices and \(E \subseteq V \times V = \{(i, j) : i, j \in V\}\) is the set of edges. Given a one-to-one labeling function \(\varphi : V \to \{1, 2, \ldots, n\}\), called a linear arrangement, the total edge length (cost) for \(G\) with respect to the arrangement \(\varphi\) is defined according to

\[
LA(G, \varphi) = \sum_{(u, v) \in E} |\varphi(u) - \varphi(v)|. 
\]

Then the MinLA problem consists in finding a best labeling function \(\varphi\) for a given graph \(G\) so that \(LA(G, \varphi)\) is minimized.
There exist polynomial time exact algorithms for some special cases of MinLA such as trees, rooted trees, hypercubes, meshes, outerplanar graphs, and others (see [3] for a detailed survey). However, as is the case with many graph layout problems, finding the minimum linear arrangement is known to be NP-hard for general graphs [5]. Therefore, there is a need for heuristics to address this problem in reasonable time. Among the reported algorithms are (a) heuristics especially developed for MinLA, such as the improved frontal increase minimization (IFIM) heuristic [6], the binary decomposition tree (BDT) heuristic [7], the multi-scale (MS) algorithm [8] and the algebraic multi-grid (AMG) scheme [9]; and (b) metaheuristics such as simulated annealing (SA) [10] and memetic algorithms (MA) [11].

In this paper, we present a highly effective two-stage simulated annealing (TSSA) algorithm for the MinLA problem. This new algorithm integrates several important features such as an efficient heuristic to generate good quality initial solutions, a highly discriminating evaluation function, a special neighborhood function and an effective cooling schedule. The performance of this algorithm is assessed with a set of benchmark instances taken from the literature. The computational results are reported and compared with the previously published ones, showing that our algorithm is able to improve the previous best-known results for 17 out of 30 instances. The influences of some key elements of the proposed SA are empirically studied and analyzed.

The rest of the paper is organized as follows. In Section 2, a brief review is given to present six most representative solution procedures for the MinLA problem. Then the components of our TSSA algorithm are discussed in detail in Section 3. Section 4 is dedicated to computational experiments and comparisons with previous results. Influences of some important components in the proposed algorithm are discussed and analyzed in Section 5. The last section summarizes the main contributions of the work.

2. Relevant existing procedures

Because of the practical and theoretical significance of the MinLA problem, much research has been carried out in developing effective heuristics for it. This is the case of the SS + SA heuristic proposed by Petit [12]. This algorithm consists in obtaining an initial solution using the spectral sequencing (SS) method [13]. Then, the resulting arrangement is locally improved through the SA algorithm reported in [10]. This SA algorithm, which implements a geometric cooling schedule, is based on a special neighborhood distribution that tends to favor moves with high probability to be accepted. The author makes computational comparisons of the SS procedure, the SA algorithm and the combination of both methods over a set of 21 benchmark graphs. He concludes that the SS + SA heuristic always improves the SS solutions and only for two graphs (e5y and gd96a) it is unable to improve the SA solution. The SS + SA running times are usually lower than those of SA.

Besides Petit’s work, Bar-Yehuda et al. present in [7] a divide-and-conquer approach to the MinLA problem. They have developed a polynomial time algorithm (with complexity $O(|V|^{2.2})$) for computing a linear arrangement induced by a binary decomposition tree (BDT). To assess their approach, the authors used the same test-suite proposed in [10] by Petit. They applied their algorithm iteratively, starting each iteration with the result of the previous one. After a few tens of iterations, the algorithm usually yields results within $5–10\%$ of those obtained by Petit’s SA [10], but at a fraction of its running time. They have used these computed arrangements as an initial solution for the SA reported in [10] and slightly better results were obtained.

In 1999, a linear time heuristic (with complexity $O(|E|)$) based on frontal increase minimization (FIM) was developed by McAllister [6]. In this paper the author compares his improved FIM heuristic (IFIM) with four existing bandwidth and profile reduction algorithms (Reverse Cuthill–McKee, FIM, Gibbs–Poole–Stockmeyer, Gibbs–King) and one MinLA algorithm (Eigenvalue-based method) over 34 benchmark instances collected by himself. The benchmarks are divided into two sets. One with 20 graphs derived from software diagrams and the other composed of 14 structure problems from the Rutherford–Boeing sparse matrix collection. He concludes that the IFIM algorithm provides the best arrangement for 17 graphs of the first set. While for the structure problems it provides superior performance in 14 graphs, compared with the four bandwidth and profile reduction algorithms. However, in comparison with the eigenvalue-based approach his algorithm is less competitive since it returns a better solution only in five cases.

Koren and Harel presented in 2002 another linear time algorithm for the MinLA problem, based on the combination of spectral methods and the MS paradigm [8]. MS techniques transform a high-dimensional problem in an iterative fashion into subproblems of increasingly lower dimensions, via a process called coarsening. On the coarsest scale the problem is solved exactly, following which a refinement process starts, whereby the solution is progressively projected back into higher and higher dimensions, updated appropriately at each scale, until the original problem is reproduced
and solved. This set of steps is called a V-cycle. The authors have also used the test-suite provided by Petit [10]. For each graph in this set, they ran their MS algorithm first with a single V-cycle and then with 10. The quality of their results after 10 V-cycle iterations is comparable to that of Petit's SA [10], but the running time is significantly lower.

In 2004, an improvement to the MS algorithm, called the Algebraic Multi-Grid scheme (AMG), was presented by Safr et al. [9]. The main difference between these approaches is the coarsening scheme. MS uses strict aggregation, while AMG employs weighted aggregation. In a strict aggregation procedure the nodes of the graph are blocked into small disjoint subsets, called aggregates. In contrast, in the weighted aggregation each node can be divided into fractions, and different fractions belong to different aggregates. Safr et al. have shown experimentally that their approach (AMG) can obtain high quality results in linear time for the MinLA problem and can be considered as one of the best MinLA algorithms known today.

In [11] we have presented a MA for finding near optimum solutions for the MinLA problem. This algorithm incorporates a highly specialized crossover operator, a fast MinLA heuristic used to create the initial population and a local search operator based on a fine-tuned SA algorithm. Later, a refined evaluation function was incorporated to our MA in [14], which provides an effective guidance in the search process. The performance of our improved MA was assessed through extensive experimentation over the test-suite presented in [10]. The results obtained were compared with the previously published ones, and showed that our MA is competitive in terms of solution quality. It was able to improve on eight previous best-known solutions and to equal these results in eight more instances. However, given that it is a MA, it consumes considerably more computer time than some heuristics specially developed for MinLA such as BDT [7], MS [8] and AMG [9].

3. A two-stage simulated annealing algorithm

SA is a general-purpose stochastic optimization technique that has proved to be an effective tool for approximating globally optimal solutions to many NP-hard optimization problems. It generally has only one significant disadvantage, its typically very long computation times.

Accelerating the SA algorithm has been an active area of research since its introduction in 1983. Most studies have concentrated on the development of faster cooling schedules [15–17], alternative move generation and acceptance strategies [18], noisy cost evaluation [19], and optimal finite-time temperature schedules [20–22]. Another approach, and the one we consider here, is TSSA [23–26].

In a TSSA algorithm a faster heuristic is used to replace the SA actions occurring at the highest temperatures. The heuristic is then followed by an improving process based on a conventional SA initiated at a temperature lower than the normal. The principal consideration in the design of a TSSA system is the determination of the starting temperature for the SA phase.

In this section we present a new TSSA implementation for solving the MinLA problem. This TSSA has the merit of improving four key features that have a great impact on its performance: an efficient heuristic to generate good quality initial solutions, a highly discriminating evaluation function called \( \Phi \), a special neighborhood function and an effective cooling schedule. Next, all the details of the implementation proposed, called TSSA-\( \Phi \), are presented.

3.1. Internal representation of linear arrangements

A graph \( G = (V, E) \) is given with vertex set \( V \) (\( |V| = n \)) and edge set \( E \). A linear arrangement \( \phi \) is represented as an array \( l \) of integers with length \( n \), which is indexed by the vertices and whose \( i \)th value \( l[i] \) denotes the label assigned to the vertex \( i \).

3.2. Neighborhood function

The search space \( \mathcal{S} \) for the MinLA problem is composed of all possible arrangements from \( V \) to \( \{1, 2, \ldots, n\} \). It is easy to see then, that there are \( n!/2 \) possible linear arrangements for a graph with \( n \) vertices.\(^1\) Next, we present some preliminary concepts used in the definition of a suitable neighborhood function for this search space.

\(^1\) Because each one of the \( n! \) arrangements can be reversed to obtain the same cost.
Let us define the partial cost contribution $L(u, \varphi)$ of a vertex $u$ with respect to the linear arrangement $\varphi$ as follows:

$$L(u, \varphi) = \sum_{v \in A(u)} |\varphi(u) - \varphi(v)|,$$

where $A(u)$ is the set of adjacent vertices of $u$. Let $\text{swap}(\varphi, u, v)$ be a function allowing to exchange the labels of two vertices $u$ and $v$ from the arrangement $\varphi$. Given that our goal is to swap labels in the graph to reduce the current value of $L(u, \varphi)$, we have observed that the best way for labeling a vertex can be found by using the following definition. Given a vertex $u$ with $d$ adjacent vertices $A(u) = \{v_1, v_2, \ldots, v_d\}$, sort them so that $s_1 < s_2 < \cdots < s_d$, where $s_i$ is called the $i$th order statistic [27]. Then the statistical median of the labels currently assigned to the vertices in $A(u)$ is given by

$$\text{median}(u) = \begin{cases} s_{(d+1)/2} & \text{if } d \text{ is odd,} \\ \frac{1}{2}(s_{(d/2)} + s_{(1+d/2)}) & \text{if } d \text{ is even.} \end{cases} \quad (2)$$

So we can construct a set $M(u)$ of vertices whose current labels are close to $\text{median}(u)$. Among the labels of those vertices in $M(u)$, we can find the best choice for labeling $u$ with respect to its adjacent vertices. $M(u)$ can be formally defined as follows: $M(u) = \{v : \text{median}(u) - 2 \leq \varphi(v) \leq \text{median}(u) + 2\}$.

This concept is better understood if we illustrate it with an example. Consider the subgraph in Fig. 1, taken from a graph with 30 vertices. This subgraph represents the vertex $u$ and its five adjacent vertices. The current labeling is given by the numbers shown inside each vertex and its current partial cost contribution is $L(u, \varphi) = 70$. Using Eq. (2) we obtain $\text{median}(u) = 26$, so $M(u)$ contains those vertices $v$ that currently have a label between 24 and 28 (some of them are not shown in Fig. 1). By examining each of these five possible labels for $u$, we can observe that the best choice is either the label 25 or 27. Both of them reduce the partial cost contribution of $u$ from 70 to 51. Remark that the bigger the absolute difference between a label and $\text{median}(u)$ is, the bigger the partial cost contribution of $u$ will be. For example, if label 1 is assigned to vertex $u$ the maximal value of $L(u, \varphi)$ is obtained (87), because label 1 produces the bigger absolute difference with respect to $\text{median}(u)$ in the graph (25).

Now, once this point is clarified, we can define the neighborhood $N_1(\varphi)$ of a labeling $\varphi$ in our TSSA implementation such that

$$N_1(\varphi) = \{\varphi' \in \mathcal{A} : \text{swap}(\varphi, u, v) = \varphi', (u, v) \in V, v \in M(u)\}. \quad (3)$$

We have observed, from the experiments presented in Section 5.1, that the $N_1(\varphi)$ neighborhood function allows to quickly reduce the total edge length of a graph; however, it presents a potential disadvantage. Given its exploitation power, it causes that our TSSA algorithm gets stuck longer on some local minima. So we have decided to combine it with another neighborhood function, with complementary characteristics, in order to get a better commitment between exploration and exploitation power. The second neighborhood function $N_2(\varphi)$ is defined as follows:

$$N_2(\varphi) = \{\varphi' \in \mathcal{A} : \text{swap}(\varphi, u, v) = \varphi', (u, v) \in V, u \neq v\}. \quad (4)$$

During the search process a combination of both $N_1(\varphi)$ and $N_2(\varphi)$ neighborhood functions is used. The former is applied with probability $p$, while the latter is employed at a $(1 - p)$ rate. This combined neighborhood function $N_3(\varphi, x)$ is defined in Eq. (5), where $x$ is a random number in the interval $[0, 1]$.

$$N_3(\varphi, x) = \begin{cases} N_1(\varphi) & \text{if } x \leq p, \\ N_2(\varphi) & \text{if } x > p. \end{cases} \quad (5)$$
3.3. Evaluation function

The evaluation function is one of the key elements for the success of heuristic search algorithms. It is the evaluation function that guides the search process toward good solutions in a combinatorial search space. The more informative this function is, the more effective the search process will be.

In combinatorial optimization, the objective function associated with a particular problem is often used as an evaluation function. However, this method cannot be used if the search space includes infeasible solutions. In such cases, penalty terms are often added to evaluate the degree of infeasibility [28]. It is also effective to dynamically change the evaluation function during the search, like in the noise method [29] and the search space smoothing method [30]. Another technique consists in developing new more informative evaluation functions which may not be directly related to the objective function such in [31].

The algorithms previously developed to solve the MinLA problem have a point in common, all of them evaluate the quality of a solution (linear arrangement) as the change in the objective function \( LA(G, \varphi) \) (Eq. (1)). A particular resulting value of the LA evaluation function can also be expressed by Formula (6), where \( d_k \) refers to the number of absolute differences with value \( k \) between adjacent vertices of the graph:

\[
LA(G, \varphi) = \sum_{k=1}^{n-1} kd_k. \tag{6}
\]

However, using LA as the evaluation function of a search algorithm represents some potential drawbacks. Indeed, LA is not sensitive enough to locate promising search regions on the space of solutions, because it does not make distinctions among the number of absolute differences \( d_k \). In other words, LA considers exactly equal a big absolute difference and a small one. Additionally, it is not really prospective because when two arrangements have the same cost it is impossible to know which one has higher possibility for further improvement. This point will be made clear below.

In this TSSA implementation we have decided to use the refined \( \Phi \) evaluation function presented in [14], which allows to overcome these disadvantages. This function evaluates the quality of an arrangement considering not only the total edge length \( LA \) of the arrangement, but also additional information induced by the number of absolute differences with value \( k \) between adjacent vertices of the graph \( d_k \). Furthermore, it maintains the fact that \( |\Phi| \) results into the same integer value produced by Eqs. (1) and (6).

The main idea of \( \Phi \) is to penalize the absolute differences \( d_k \) having small values of \( k \) and to favor those with values of \( k \) near to the bandwidth \( \beta \) of the graph.\(^2\) The logic behind this is that it is easier to reduce the total edge length of the arrangement if it has summands of greater value. To accomplish it, each number of absolute differences \( d_k \) should have a different contribution, which is computed by employing

\[
k + \frac{1}{\prod_{j=1}^{k} (n + j)} = k + \frac{n!}{(n + k)!}. \tag{7}
\]

Then, by combining Formulas (6) and (7) we obtain Eq. (8). This formula can be used to compute the quality of an arrangement and represents the \( \Phi \) evaluation function. Observe that the first term in this formula is equal to Eq. (6). The second term (a fractional value) is the discriminator for arrangements having the same LA value:

\[
\Phi(G, \varphi) = \sum_{k=1}^{n-1} \left( k + \frac{n!}{(n + k)!} \right) d_k = \sum_{k=1}^{n-1} kd_k + \sum_{k=1}^{n-1} \frac{n!d_k}{(n + k)!}. \tag{8}
\]

The choice of \( \Phi \) as evaluation function is fully justified by the fact that it is more discriminating than LA and leads to smoother landscapes of the search process, as it was demonstrated by the experimental results presented in [14] and confirmed by those described in Section 5.2. But also because \( \Phi \) allows an incremental cost evaluation of neighboring solutions if each term \( k + (n!/(n + k)!)) \) in the Eq. (8) is precalculated and stored in an array. Suppose for example that the labels of two different vertices \( (u, v) \) are swapped, as in the neighborhood functions presented in Section 3.2, then only the \( |A(u)| + |A(v)| \) absolute differences that change should be recomputed to update the value of \( \Phi \). It is faster than the \( O(|E|) \) instructions originally required.

\(^2\) \( \beta(G, \varphi) = \max\{|\varphi(u) - \varphi(v)| : (u, v) \in E\} \)
3.4. Initial solution

The initial solution is the starting labeling used for the algorithm to begin the search for better configurations in the search space \( \mathcal{A} \). After a comparison, both in solution quality and computation time, of the different existing heuristics for solving MinLA we have decided to use the method proposed by McAllister in [6]. This decision is based on two points: the high solution quality produced by the heuristic and its small computational time.

The heuristic proposed by McAllister is a vertex-by-vertex greedy algorithm based on the following two basic steps:

1. Select a starting vertex and place it in position 1. 
2. For each remaining vertex \( 2 \) through \( n \), select one of the unplaced vertices for placement in the current position by using the FIM strategy. It consists in selecting for placement \( i \) a vertex that is adjacent to the fewest vertices in \( U_i - F_i \), where \( F_i = \{ u \in U_i : v \in P_i \text{ and } (u, v) \in E \} \) denotes the \textit{front} at placement \( i \), \( P_i \) represents the set of \( i - 1 \) vertices placed so far and \( U_i \) the set of currently unplaced vertices.

In order to accomplish this, McAllister has defined two measures, that enable to know how highly a vertex \( v \in U_i \) is connected to \( P_i \) and to \( U_{i+1} \). They are defined respectively as follows: 
\[
\ell_i(v) = \{ (u, v) \in E : u \in P_i \} \\
tr_i(v) = d(v) - \ell_i(v),
\]
where \( d(v) \) denotes the degree of the vertex \( v \). Both measures are used to define a new selection factor \( sf_i(v) = tr_i(v) - \ell_i(v) \), which is used at the two-step general strategy described above as follows: For each placement \( i \) in step 2, select \( v \in P_i \) with minimum \( sf_i(v) \). The proposed algorithm has a linear time complexity with respect to the number of edges in the graph. This is possible thanks to the use of efficient data structures that enable to select a vertex with minimum \( sf_i(v) \) in constant time.

3.5. Initial temperature determination

We have decided to initialize the temperature for the TSSA-\( \Phi \) using the method proposed by Varanelli and Cohoon in [26]. Their work is based on large-scale numerical studies, conducted by different authors [21,32–34], which examine solution densities at varying SA temperatures. These investigations present evidence that supports a typical behavior of the expected cost \( C_k \) and standard deviation \( \sigma_k \) with respect to SA temperature \( T_k \) given homogeneous SA cooling schedule. Additionally, these studies independently show that the probability distribution of the cost values can be closely approximated by a normal distribution which presents the following behaviors:

\[
C_k \approx C_\infty - (\sigma_\infty^2 / T_k),
\]
\[
\sigma_k \approx \sigma_\infty,
\]
where \( C_\infty \) and \( \sigma_\infty \), respectively, represent the expected cost and the standard deviation of the cost over the solution space.

Given this behavioral information Varanelli and Cohoon have proposed an equation that permits to approximate the SA temperature \( T_k(i) \) at which a solution \( i \) with cost \( c(i) \) would be found as the best-so-far solution:

\[
T_k(i) \approx \frac{\sigma_\infty^2}{C_\infty - c(i)} - \gamma_\infty \sigma_\infty.
\]

In Eq. (11), the parameter \( \gamma_\infty \) represents the offset between the expected cost \( C_k \) and the best-so-far solution cost \( c(i) \) at the temperature \( T_k \). It can be calculated probabilistically by using the following expression, where \( r \) denotes the number of moves generated at each temperature:

\[
P[C_\infty - \gamma_\infty \sigma_\infty < X < C_\infty + \gamma_\infty \sigma_\infty] \approx 1 - |r|^{-1}.
\]

The reader is referred to [26] for a detailed explanation of the equations derivation and proofs.

In our implementation, we have proceeded as follows: First, \( 10^3 \) independent random solutions are generated, with the mean and standard deviation of the cost values recorded. These values then serve as approximations for the expected cost over the solution space \( C_\infty \) and the standard deviation of cost over the solution space \( \sigma_\infty \). Next, a heuristic solution with cost \( c(i) \) is obtained using the method described in Section 3.4. Then, the offset \( \gamma_\infty \) between the expected cost \( C_k \) and the best-so-far solution cost \( c(i) \) is computed by using Eq. (12). Finally, the values \( C_\infty, \sigma_\infty, c(i) \) and \( \gamma_\infty \) are used in Eq. (11) to obtain the starting temperature approximation \( T_k(i) \) that will be used in the second stage of the TSSA-\( \Phi \) algorithm.
3.6. Cooling schedule

The cooling schedule determines the degree of uphill movement permitted during the search and is, thus, critical to the algorithm’s performance. The literature offers several cooling schedules, see for instance those proposed in [15,21,16,26]. In the TSSA-Φ implementation we preferred the statistical cooling schedule proposed in [15] because it has demonstrated to be very effective [32,35] in several combinatorial optimization problems.

In our implementation the statistical cooling schedule starts at the initial temperature approximation \( T_0 \) computed with Formula (11). Then, at each round, decrements the current temperature by using the following relation:

\[
T_k = T_{k-1} \left( 1 + \frac{\ln(1 + \delta) T_{k-1}}{3 \sigma_{T_{k-1}}} \right)^{-1},
\]

where \( \sigma_{T_{k-1}} \) is the standard deviation of the evaluation function values at the current temperature and \( \delta \) is called the distance parameter. Small \( \delta \) values lead to small temperature decrements. For each temperature, the maximum number of generated neighboring labelings is \( r \), it depends directly on the number of edges \( |E| \) of the graph. This is because more moves are required for denser graphs.

3.7. Termination condition

The algorithm stops when the evaluation function mean value shows only very small changes. In practice it is achieved by computing the derivative of the smoothed evaluation function mean value \( \overline{SC}_k \). Then the algorithm terminates if at certain temperature \( T_k \) the condition \( \overline{SC}_k < \varepsilon \) is met. We call \( \varepsilon \) the stop factor and it is a small positive value.

4. Computational experiments

In this section, we present a set of experiments accomplished to evaluate the performance of the TSSA-Φ algorithm presented in Section 3. For these experiments, the algorithms were coded in C and compiled with gcc using the optimization flag -O3. They were run sequentially into a CPU Xeon at 2 GHz, 1 GB of RAM and Linux. Due to the incomplete and non-deterministic nature of the algorithms, 10 independent runs were executed for each of the selected benchmark instances. When averaged results are reported, they are based on these 10 corresponding executions. In all the experiments the following parameters were used for TSSA-Φ:

(a) Initial temperature \( T_0 \) computed with the procedure described in Section 3.5.
(b) Cooling schedule distance parameter \( \delta = 0.10 \).
(c) Maximum neighboring solutions per temperature \( r \):

\[
r = \begin{cases} 
5.0E+05 & \text{if } 1 < |E| \leq 500, \\
2.0E+06 & \text{if } 501 < |E| \leq 50,000, \\
3.5E+06 & \text{if } 50001 < |E| \leq 1.1E+06, \\
7.0E+06 & \text{if } |E| > 1.1E+06. 
\end{cases}
\]

(d) Stop factor \( \varepsilon = 1.0E + 10 \).
(e) The neighborhood function \( N_3(\varphi, x) \) is applied using a probability \( p = 0.90 \).

4.1. Benchmark instances and comparison criteria

The test-suite that we have used in the experiments is divided into two subsets. The first subset consists of the 21 benchmarks\(^3\) proposed in [10] and used later in [7–9,11,14]. It includes graphs from six different families: uniform random, geometric random, graphs with known optima, finite element discretizations, VLSI design and graph drawing competitions. Their number of vertices is between 62 and 9800. The second subset is composed of nine very large

\(^3\)WWW.LSI.UPC.ES/~jPETIT/MinLA/Experiments
graphs from finite element discretizations, obtained from the publicly available collections of George Karypis\(^4\) and Francois Pellegrini.\(^5\) These graphs were first used by Koren and Harel [8] and later by Safro et al. in [9]. Their number of vertices is between 78136 and 1017253.

To assess the performance of our TSSA-\(\Phi\), we show comparative results on these benchmark instances. The main criterion used for the comparison is the same as the one commonly used in the literature: the best total edge length found (smaller values are better). Computing time is also given for indicative purpose.

The comparison is carried out in two parts. The first part gives an instance-by-instance performance comparison of our TSSA-\(\Phi\) algorithm with respect to the state-of-the-art heuristics, using the test-suite proposed in [10]. In the second part TSSA-\(\Phi\) is compared against two previously reported best algorithms over 9 very large graphs [8].

4.2. Comparing TSSA-\(\Phi\) with the state-of-the-art algorithms

The purpose of this experiment is to carry out a performance comparison of our TSSA-\(\Phi\) algorithm, presented in Section 3, with respect to the following well-known heuristics: SS+SA [10,12], BDT+SA [7], MS [8], AMG [9] and MA [14].

Table 1 displays the detailed computational results produced by this experiment. The first three columns in the table indicate the name of the graph, its number of vertices and its number of edges. The next column presents the initial solutions used by our TSSA-\(\Phi\) algorithm, which were generated with the IFIM [6] heuristic detailed in Section 3.4. IFIM is very fast and it is able to compute a good quality solution for the largest graph in this test-suite (whitaker3) in 0.18 s. Column 5 shows the previous best-known solution reported in the literature for each of the studied instances, while column 6 indicates the reference where this result was obtained. Next four columns present the best cost in terms of total edge length (\(C\)), the average cost (\(\text{Avg.}\)), its standard deviation (\(\text{Dev.}\)) and the average CPU time (\(T\)) in seconds.

\(^4\) ftp.cs.umn.edu/users/kumar/Graphs
\(^5\) www.labri.u-bordeaux.fr/Equipe/PARADIS/Member/pelegrin/graph

---

Table 1

Performance comparison between TSSA-\(\Phi\) and the following state-of-the-art heuristics: SS + SA [10,12], BDT + SA [7], MS [8], AMG [9] and MA [14]; over 21 benchmarks proposed in [10]

| Graph  | \(|V|\) | \(|E|\) | IFIM | \(C\) | Reference | TSSA-\(\Phi\) | \(\Delta C\) |
|--------|-------|-------|------|-----|----------|------------|----------|
|        |       |       |      |     |          | C          | Avg.     |
|        |       |       |      |     |          | Dev.       | T        |
| randomA1 | 1000  | 4974  | 972 583 | 867 214 | [14] | 866 968  | 866 975 4 |
| randomA2 | 1000  | 24 738 | 6 724 470 | 6 528 780 | [10] | 6 522 206 | 6 522 221 6 |
| randomA4 | 1000  | 8177  | 1 845 977 | 1 718 746 | [14] | 1 717 176 | 1 717 179 6 |
| randomG4 | 1000  | 8173  | 280 961 | 140 211 | [9,14] | 140 596  | 140 597 0 |
| bintree10 | 1023  | 1022  | 71 517 | 3696 | [9] | 3696  | 3697 1 |
| bc10    | 1024  | 5120  | 523 776 | 523 776 | [7–9,14] | 523 776  | 523 776 0 |
| mesh33x33 | 1089  | 2112  | 44 430 | 31 729 | [8,9] | 31 856  | 31 904 2 |
| 3lc    | 4720  | 13 722 | 481 815 | 357 329 | [9,14] | 359 151  | 359 176 0 |
| airfoil1 | 4253  | 12 289 | 384 013 | 272 931 | [9] | 276 381  | 276 866 5 |
| whitaker3  | 9800  | 28 989 | 1 231 912 | 1 144 476 | [9] | 1 143 645 | 1 145 304 7 |
| c1y    | 828   | 1749  | 70 922 | 62 262 | [9,14] | 62 230  | 62 234 4 |
| c2y    | 980   | 2102  | 90 347 | 78 822 | [9,14] | 78 757  | 78 810 8 |
| c3y    | 1327  | 2844  | 151 622 | 123 376 | [14] | 123 145  | 123 151 1 |
| c4y    | 1366  | 2915  | 131 106 | 115 051 | [14] | 114 936  | 114 971 6 |
| c5y    | 1202  | 2557  | 118 541 | 96 878 | [14] | 96 850  | 96 877 2 |
| gd95c  | 62    | 144   | 525 506 | [7,9,14] | 506  | 506 1  |
| gd96a  | 1096  | 1676  | 146 407 | 95 242 | [14] | 95 263  | 95 277 9 |
| gd96b  | 111   | 193   | 1497 1416 | [9,14] | 1416  | 1417 8 |
| gd96c  | 65    | 125   | 537 519 | [7–10,14] | 519  | 519 1  |
| gd96d  | 180   | 228   | 2937 2391 | [9,14] | 2391  | 2394 2  |

Average 1.257E+06 1.255E+06 2286.5
obtained in 10 executions of the TSSA-Φ algorithm. The quantities in column $T$ include the time expended by the initial solution computation. Finally, the difference ($A_C$) between the best cost produced by TSSA-Φ and the previous best-known solution is depicted in the last column.

From Table 1, one observes that TSSA-Φ is very competitive in terms of solution quality, because it obtains a better average solution quality (1.255E + 06) than the best-known solutions (1.257E+06) provided by the five compared methods. Indeed, TSSA-Φ is able to improve on 10 previous best-known solutions and to equal these results in six instances. In several benchmarks the improvement achieved leads to a significant decrease of the cost ($A_C$ up to −44 129). For the other five tested graphs, TSSA-Φ did not reach the best reported solutions, but its results are very close to them (in average 0.494%). On the other hand, the SS + SA, BDT + SA, MS, AMG and MA algorithms only equal the previous best-known solutions in 2, 3, 3, 13 and 15 instances, respectively. These five algorithms have found arrangements with higher cost than the previous best-known solutions for the rest of the 21 graphs.

With respect to the computational effort we would like to point that, the running times from these five algorithms cannot be compared directly with ours, since the computational platforms that were used are different. Nevertheless, to have an idea of these magnitude differences, we have obtained a CPU comparison made with 3300 benchmarks from the Internet site Tom’s hardware guide.\footnote{\url{www.tomshardware.fr/cat.php?c=21,histoire et hit parade des CPU.}}

This information allows us to observe that the running time of SS+SA for the largest graph in this test-suite (whitaker3) was over 11 h using a computer (K6 III at 600 MHz) which is about 5 times slower than our platform. For the same instance and compared with our computer, the CPU time consumed for each of the other algorithms is: BDT+SA, 8 h using a machine (Pentium III at 600 MHz) 3 times slower; MS, 127.79 s in a CPU (Pentium III at 700 MHz), which is about 2.6 times slower; MA, 15299.72 s employing a computer with the same characteristics than ours. For the AMG algorithm the authors did not present the consumed CPU time, so we have obtained their source code to compile it in a computer similar to that used in [9] (Pentium IV at 1.7 GHz), which is approximately 0.2% slower than ours. Then, it was executed on the graph whitaker3 resulting in a CPU time of 280.28 s.

In conclusion, even if the results attained by our TSSA-Φ algorithm are very competitive, we have observed that it consumes slightly more CPU time than some heuristics specially developed for MinLA such as MS [8] and AMG [9]; but our algorithm is much faster than the previous proposed evolutionary approach MA [14]. Compared with the other two existing SA based algorithms (SS + SA and BDT + SA) our approach consumes in average a computing time slightly smaller than that expended by them.

4.3. Comparing TSSA-Φ on very large graphs

There are only two MinLA heuristics previously reported in the literature that present experiments with the very large instances proposed by Koren and Harel: MS [8] and AMG [9]. In this section we present a comparison of the results attained by the TSSA-Φ algorithm, over this test-suite [8], with respect to those obtained by the MS and AMG heuristics.

The results presented in Section 4.2 show that TSSA-Φ consumes slightly more CPU time than the MS and AMG heuristics. Then, to make a fair comparison among the three methods, we have decided to divide it into two parts using two different termination conditions. One stopping the TSSA-Φ algorithm immediately after it begins to improve the previous best-known solution and the other one using the usual termination condition presented in Section 3.7.

The data obtained in these comparisons are compiled in Tables 2 and 3, respectively. Both tables present: in the first column, the name of the studied instance; in the penultimate column, the difference between the best cost attained by TSSA-Φ, using the corresponding stop condition, and the previous best-known solution ($A_C$); and in the last column the ratio of the CPU time consumed by our algorithm and the heuristic which has found the previous best-known solution ($A_T$). Columns titled $C$ and $T$ (in the two tables) display the best cost attained by each one of the methods (MS, AMG, IFIM and TSSA-Φ), and the CPU time in minutes consumed to obtain it. The data in columns 8–11 from Table 2 were omitted, for the instances m14b and auto, because TSSA-Φ did not improve the previous best-known solution for these two benchmarks. The results presented for the MS and AMG heuristics were taken from their corresponding paper. In both cases the heuristics were executed in a CPU at 1.7 GHz, which is only 0.2% slower than our computer. Additionally, the average cost ($Avg.$) obtained in 10 executions of the TSSA-Φ algorithm and its standard deviation ($Dev.$) are presented in columns 5 and 6 of Table 3 for informative purpose.
Analyzing the data presented in Tables 2 and 3 lead us to the following main observations. First, the solution quality attained by the TSSA-Φ algorithm, that is stopped when it improves the previous best-known solution, is very competitive with respect to that produced by the state-of-the-art heuristics (MS and AMG). In fact, it ameliorates the previous best-known solution in seven out of nine instances, achieving an average improvement of $-2.00E+04$ (see column $ΔC$ in Table 2). For certain benchmarks, like the graph mrngA, an important reduction in cost is accomplished by our algorithm ($ΔC = -5.78E+04$). Nevertheless, TSSA-Φ finds higher cost solutions than the AMG heuristic in two of the studied graphs. For instance, the ratio of the solution found by our algorithm and that produced by AMG is 0.15% over the benchmark m14b, but it is higher (4.61%) over the graph auto producing a $ΔC = 1.78E+08$. We believe that it is due to the high graph density.

Second, the average computing time consumed by our approach, to produce these excellent results, is slightly greater than that used by the MS and AMG heuristics (in average 4.08 times greater). However, since TSSA-Φ outperforms the other two compared methods in terms of cost, we believe that the extra consumed computing time is fully justified. Furthermore, it can be observed in the second part of the comparison (Table 3) that TSSA-Φ is able to continue improving the solution quality when the stop condition described in Section 3.7 is used (i.e. it is executed for longer time). This leads to accomplish an important reductions in cost, as is the case of the graph mrngB, where the reduction obtained is $ΔC = -7.59E+07$. In contrast the MS and AMG heuristics do not take advantage of longer executions, as it was mentioned by their respective authors [8, 9].

This favorable behavior of TSSA-Φ is illustrated in Fig. 2, where its convergence process is presented over the tooth instance. The plot represents the average solution quality obtained in 10 executions (ordinate) with respect to
the consumed computing time in minutes (abscissa). Two lines denoting the best solution found by MS and AMG are also included for comparison purposes. From this figure it can be seen that the TSSA-Φ continues reducing the cost almost continuously after outperforming the other compared heuristics. For instance, it is able to reach a cost of 223,988,122.2 spending only 5.34 longer time than the AMG heuristic (i.e. 171.10 min), which represents an improvement of $\Delta C = -3.65E - 06$. The maximal amelioration in cost (214,884,320.5 vs. 227,639,682 for AMG) is obtained using 361.56 min.

5. Discussion

The purpose of the experiments presented in this section is to better understand the influences of some key features of any SA algorithm. In some of these experiments we have used a slightly modified version of the TSSA-Φ algorithm presented in Section 3, which starts from a random initial solution and at a temperature which warranties 70% of accepted moves in the first Metropolis round. This modified algorithm called OSSA (for one stage SA) allows us to better appreciate the studied influences.

The 30 benchmark instances described in Section 4.1 were used consistently over all the experiments presented in the following subsections. Similar results were obtained with all of them, so, for the reason of space limitation, we have decided to show the product of these experiments with only some representative graphs. All the results presented in this section are based on average data obtained in 10 independent runs.

5.1. Influence of the neighborhood functions

The neighborhood function is one of the critical elements for the performance of any local search algorithm. In this study we have considered not only the functions $N_1(\varphi)$ and $N_2(\varphi)$ presented in Section 3.2, but also two other neighborhood relations defined in Eqs. (15) and (16), where $\text{swap}(\varphi, u, v, w)$ is the result from applying the operations $\text{swap}(\varphi, u, v)$ and $\text{swap}(\varphi, v, w)$ successively:

$$N_4(\varphi) = \{ \varphi' \in \mathcal{A} : \text{swap}(\varphi, u, v) = \varphi', (u, v) \in V, \ v \in A(u) \},$$

$$N_5(\varphi) = \{ \varphi' \in \mathcal{A} : \text{swap}(\varphi, u, v, w) = \varphi', (u, v, w) \in V, \ u \neq v \neq w \}. \tag{16}$$

Experiments have been carried out to compare the performance of these four neighborhood functions using the OSSA algorithm (see Section 5). The plot presented in Fig. 3(a) shows the differences in terms of average solution quality attained by OSSA, when each one of the studied neighborhood relations is used to solve the tooth benchmark instance. From this graph it can be observed that the best performance is attained by OSSA when the $N_2(\varphi)$ neighborhood function is used ($LA = 235,301,805.89$). On the other hand, $N_1(\varphi)$ allows to reduce the total cost of the graph faster than $N_2(\varphi)$, however, it causes that our OSSA algorithm gets stuck on some local minima, given its exploitation power.
Taking into account the complementary characteristics of both neighborhood functions, we have decided to combine them to get a better commitment between exploration and exploitation of the search space. This combination, called \( N_3(\phi, x) \), was presented in Formula (5), where \( p \) represents the probability to apply \( N_1(\phi) \) (the rest of the time \( N_2(\phi) \) is employed).

In order to find the most suitable value for the probability \( p \), used in \( N_3(\phi, x) \), we proceeded as follows: For each one of the nine values of \( p \), in the interval \([0.10, 0.90] \) with step of 0.10, 10 executions of the OSSA algorithm over the graph tooth were performed (similar results were obtained with other instances). The average results of these executions are plotted in Fig. 3(b). It is evident from this graph that the best average cost is obtained by OSSA when a probability \( p = 0.90 \) is used by the \( N_3(\phi, x) \) neighborhood function \( (LA = 225943530.37) \). It is important to remark that this probability allows our combined \( N_3(\phi, x) \) neighborhood function to obtain even better results than \( N_2(\phi) \).

5.2. Influence of the evaluation functions

The evaluation function is in charge of guiding the search process toward good solutions in a combinatorial search space. For this reason it is a key component in any heuristic search algorithm. This subsection presents a series of experiments designed to study the characteristics of the \( \Phi \) evaluation function (see Section 3.3) and provide more insights into its real working.

First, a study of the \( \Phi \) evaluation function effectiveness with respect to the conventional \( LA \) evaluation function was conducted. For this purpose, a steepest descent (SD) algorithm was employed. The choice of the SD algorithm for this comparison is fully justified by the fact that SD is completely parameter free, and thus it allows a direct comparison of the two evaluation functions without bias. The implemented SD algorithm with evaluation function \( f(\phi) \) starts from the initial solution \( \phi \in \mathcal{A} \) and repeats replacing \( \phi \) with the best solution in its neighborhood \( N_2(\phi) \) until no better arrangement is found. Lets call SD-LA and SD-\( \Phi \) the algorithm depending on which evaluation function it uses.

The results of this comparative study were presented in detail in [14]. From these results one observes that the SD algorithm that employs \( \Phi \), consistently has better results than the algorithm that uses \( LA \) in all the tested instances. The advantage of using \( \Phi \) as evaluation function is well summarized in Fig. 4, where the behavior of the studied evaluation functions is presented over the random A1 instance.

In Fig. 4(a) the \( X \)-axis represents the number of moves, while the \( Y \)-axis indicates the average solution quality. Remark that the SD-LA algorithm stops the search process earlier than SD-\( \Phi \), basically because \( LA \) cannot distinguish arrangements with the same cost given as consequence a critical deficiency in finding improving neighbors. This fact is easily seen in Fig. 4(b) where the evolution of the average number of improving neighbors \( (Y\text{-axis}) \) with respect to the number of moves is depicted. From this figure we can observe that SD-\( \Phi \) produces better results because it is capable
After having studied the characteristics of $\Phi$ by using a simple SD algorithm, we have decided to evaluate its practical usefulness within the OSSA algorithm described in Section 5. Let us call it OSSA-LA or OSSA-\(\Phi\) to distinguish which evaluation function is employed. The algorithms were compiled and run in our computational platform using the same parameters in both cases.

In all the studied instances we have observed that OSSA-\(\Phi\) consistently produces better results than OSSA-LA. This dominance is illustrated in Fig. 5, where the behavior of the studied evaluation functions is analyzed over the randomA1 instance. The ordinate represents the solution quality, while the abscissa indicates the number of moves. From this graph it can be seen that OSSA-\(\Phi\) is more effective in searching better solutions than OSSA-LA at every instant of the search process. Indeed, it is possible because \(\Phi\) is able to overcome the disadvantages presented by the classic evaluation function (see Section 3.3).
The third experiment has the objective to understand the influence of the cooling schedule in our TSSA implementation. For this study we have selected two representative cooling schedules reported in the literature: geometrical [36] and statistical [15]. Then, we have implemented them within the OSSA algorithm described at the begin of Section 5. We named them OSSA-geometric or OSSA-statistical to distinguish which cooling schedule is employed.

For both algorithms the method described in [15] to compute the initial temperature was used. It permits to ensure a selected initial acceptance rate. The two algorithms start from a random initial solution and generate $r$ moves at each Metropolis round. Then, in the OSSA-geometric algorithm, the next temperature is calculated by reducing its current value ($T_{k-1}$) with the use of the following relation: $T_k = T_{k-1} \times 0.96$. On the other hand, OSSA-statistical decrements the current temperature by applying Eq. (13). This process continues for both algorithms until the termination condition detailed in Section 3.7 is met.

The results produced by this comparison show the advantage of using the statistical cooling schedule. This tendency is exemplified in Fig. 6, which shows the typical behavior of OSSA-statistical curve plotted against the OSSA-geometric curve. The X-axis represents the number of moves, while the Y-axis corresponds to the solution quality. This graph also includes a third curve representing the evolution of the TSSA-$\Phi$ presented in 3 for comparative purposes. Remember that TSSA-$\Phi$ uses also the statistical cooling schedule, but it starts from a good quality solution and at a suitable temperature determined by the method proposed in [26].

From Fig. 6 one observes that the OSSA algorithm using the statistical cooling schedule produces better results than those obtained by the OSSA-geometric algorithm and using less moves. Indeed, it is possible because OSSA-statistical adjusts the current temperature by using certain collected information from the visited regions of the search space, while OSSA-geometric does not. It allows OSSA-statistical to explore the search space in a more efficient way. The third curve in Fig. 6 enables us to observe the important speedup achieved by the TSSA-$\Phi$ algorithm with respect to the other two compared algorithms. The experiments carried out, over the whole test-suite, show that TSSA-$\Phi$ converges in average 24.9% faster than the OSSA-statistical algorithm, because it needs fewer moves, and consequently less computing time to return a solution of good quality.

6. Conclusion

In this paper, we have introduced a highly effective two-stage simulated annealing algorithm (TSSA-$\Phi$), which integrates the following high impact features:

- An efficient heuristic to generate initial solutions of good quality. This first stage of the search allows us to replace the SA actions occurring at the highest temperatures and thus to save important computing time.
A compound neighborhood function which combines a carefully designed neighborhood with a random swap neighborhood. This compound neighborhood allows the search to quickly reduce the total cost of a graph, while avoiding to get stuck on some local minima.

A refined and more discriminating function (Φ) to evaluate arrangements. This evaluation function considers not only the total edge length (LA) of an arrangement, but also additional semantic information contained in it to distinguish solutions with the same LA value, allowing thus the search to better explore the combinatorial space.

An effective statistical cooling schedule. It allows TSSA-Φ to take advantage of the good quality of the initial solution generated in the first stage of the algorithm, which results in a significant speedup of approximately 24.9% with respect to the traditional SA algorithm.

To assess the practical effectiveness of this TSSA-Φ algorithm, we have carried out extensive experimentation using a set of 30 benchmark instances of the literature: 21 small and medium sized instances from [10] and 9 very large instances from [8]. In these experiments the TSSA-Φ algorithm was carefully compared with five state-of-the-art algorithms. The results show that TSSA-Φ was able to improve 17 previous best-known solutions out of 30 benchmarks in terms of solution quality: 10 in the set of 21 instances and 7 in the set of 9 very large graphs.

Also, we have shown extensive studies about three key elements of the TSSA-Φ algorithm: neighborhood function, evaluation function and cooling schedule; confirming that appropriated choices of these elements are indispensable for reaching high performance of a SA algorithm.

Finally, we hope the design of the TSSA-Φ algorithm sheds some additional lights on how SA should be adapted for effective solving of hard combinational problems.

Acknowledgments

This work is supported by the CONACyT Mexico, the “Constrat Plan Etat-Région” project COM (2000–2006) as well as the Franco-Mexican Joint Lab in Computer Science LAFMI (2005–2006). The reviewers of the paper are greatly acknowledged for their constructive comments.

References


An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem

Author(s): Rodriguez-Tello, E.; Jin-Kao Hao; Torres-Jimenez, J.


Abstract: In this paper, an improved two-stage simulated annealing algorithm is presented for the minimum linear arrangement problem for graphs. This algorithm integrates several distinguished features including an efficient heuristic to generate good quality initial solutions, a highly discriminating evaluation function, a special neighborhood function and an effective cooling schedule. The algorithm is evaluated on a set of 30 well-known benchmark instances of the literature and compared with several state-of-the-art algorithms, showing improvements of 17 previous best results. [All rights reserved Elsevier].

Accession Number: 9865594

Document Type: Journal Paper

Language: English

Treatment: Theoretical or Mathematical

Controlled Indexing: graph theory; simulated annealing

Uncontrolled Indexing: simulated annealing algorithm; minimum linear arrangement problem; graphs; heuristic; discriminating evaluation function; special neighborhood function; cooling schedule

Classification Codes: A0210 Algebra, set theory, and graph theory; B0250 Combinatorial mathematics; B0260 Optimisation techniques; C1160 Combinatorial mathematics; C1180 Optimisation techniques

Author Address: Rodriguez-Tello, E.; Jin-Kao Hao; Univ. d psila Angers, Angers, France

Publisher: Elsevier Science Ltd., UK

Number of References: 36

CODEN: CMORAP

Document Number: S0305-0548(07)00067-6

DOI: 10.1016/j.cor.2007.03.001