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 Tamps., 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.

Key words: Linear Arrangement, Evaluation Function, Heuristics, Simulated Annealing

1 Introduction

The *Minimum Linear Arrangement* problem (MinLA) was first stated by Harper in [15]. His aim was to design error-correcting codes with minimal average absolute errors on certain classes of graphs. Later, in the 1970's MinLA was

* Corresponding author.
Email addresses: ertello@info.univ-angers.fr (Eduardo Rodriguez-Tello), hao@info.univ-angers.fr (Jin-Kao Hao), jtj@cinvestav.mx (Jose Torres-Jimenez).

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 [3]. MinLA arises also in other research fields like biological applications, graph drawing, software diagram layout and job scheduling [7, 23].

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 $\varphi : V \rightarrow \{1..n\}$, called a linear arrangement, the total edge length (cost) for G with respect to the arrangement φ is defined according to Eq. 1.

$$LA(G, \varphi) = \sum_{(u,v)\in E} |\varphi(u) - \varphi(v)| \tag{1}$$

Then the MinLA problem consists in finding a best labeling function φ 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 [7] 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 [9]. 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 heuristic (IFIM) [25], the Binary Decomposition Tree heuristic (BDT) [4], the Multi-Scale algorithm (MS) [22] and the Algebraic Multi-Grid scheme (AMG) [33]; and b) metaheuristics such as Simulated Annealing (SA) [28] and Memetic Algorithms (MA) [30].

In this paper, we present a highly effective improved Two-Stage Simulated Annealing 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 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 Two-Stage Simulated Annealing 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 [27]. This algorithm consists in obtaining an initial solution using the Spectral Sequencing (SS) method [20]. Then, the resulting arrangement is locally improved through the Simulated Annealing (SA) algorithm reported in [28]. This SA algorithm, that 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 ($c5y$ 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 [4] 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 [28] 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 [28], but at a fraction of its running time. They have used these computed arrangements as an initial solution for the SA reported in [28] 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 [25]. 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 5 cases.

Koren and Harel presented in 2002 another linear time algorithm for the MinLA problem, based on the combination of spectral methods and the Multi-Scale (MS) paradigm [22]. 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 [28]. For each graph in this set, they ran their MS algorithm first with a single V-cycle and then with ten. The quality of their results after 10 V-cycle iterations is comparable to that of Petit's SA [28], 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 Safro *et al.* [33]. 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. By contrast, in the weighted aggregation each node can be divided into fractions, and different fractions belong to different aggregates. Safro *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 [30] we have presented a Memetic Algorithm (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 [31], which provides an effective guidance in the search process. The performance of our improved MA was assessed through extensive experimentation over the test-suit presented in [28]. The results obtained were compared with previously published ones, and showed that our MA is competitive in terms of solution quality. It was able to improve on 8 previous best-known solutions and to equal these results in 8 more instances. However, given that it is a memetic algorithm, it consumes considerably more computer time than some heuristics specially developed for MinLA such as BDT [4], MS [22] and AMG [33].

3 A two-stage simulated annealing algorithm

Simulated Annealing (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 [2, 17, 24], alternative move generation and acceptance strategies [10], noisy cost evaluation [11], and optimal finite-time temperature schedules [5, 14, 34]. Another approach, and the one we consider here, is Two-Stage Simulated Annealing (TSSA) [12, 18, 32, 35].

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 Φ, a special neighborhood function and an effective cooling schedule. Next all the details of the implementation proposed, called TSSA-Φ, are presented.

3.1 Internal representation of linear arrangements

Given a graph $G = (V, E)$ with vertex set V ($|V| = n$) and edge set E. A linear arrangement φ 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 $\mathscr{A}$ for the MinLA problem is composed of all possible arrangements from V to $\{1, 2, ..., n\}$. It is easy to see then, that there are $n!/2$ possible linear arrangements for a graph with n vertices [1]. Next, we present

[1] because each one of the $n!$ arrangements can be reversed to obtain the same cost.

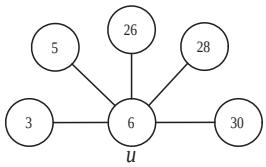


Figure 1. Labeled subgraph taken from a graph with 30 vertices.

some preliminary concepts used in the definition of a suitable neighborhood function for this search space.

Let us define the partial cost contribution $L(u, \varphi)$ of a vertex u with respect to the linear arrangement φ 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 $swap(\varphi, u, v)$ be a function allowing to exchange the labels of two vertices u and v from the arrangement φ. 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 $v_1, ..., v_d$, sort them so that $s_1 < s_2 < ... < s_d$, where s_i is called the *i-th* order statistic [16]. Then the *statistical median* of the labels currently assigned to the vertices in $A(u)$ is given by Formula 2:

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

So we can construct a set $M(u)$ of vertices whose current labels are close to $median(u)$. 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: $M(u) = \{v : median(u) - 2 \leq \varphi(v) \leq median(u) + 2\}$

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 5 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 $median(u) = 26$, 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 $median(u)$ 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 $median(u)$ in the graph (25).

Now, once this point is clarified, we can define the neighborhood $N_1(\varphi)$ of a

labeling φ in our TSSA implementation such that:

$$N_1(\varphi) = \{\varphi' \in \mathscr{A} : swap(\varphi, u, v) = \varphi',\ (u, v) \in V,\ v \in M(u)\} \tag{3}$$

We have observed, from the experiments presented in Subsection 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. The second neighborhood function $N_2(\varphi)$ is defined as follows:

$$N_2(\varphi) = \{\varphi' \in \mathscr{A} : swap(\varphi, u, v) = \varphi',\ (u, v) \in V,\ u \neq v\} \tag{4}$$

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 $[0, 1]$.

$$N_3(\varphi, x) = \begin{cases} N_1(\varphi) & \text{if } x \leq p \\ N_2(\varphi) & \text{if } x > p \end{cases} \tag{5}$$

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 to a particular problem is often used as an evaluation function. However, this method can not be used if the search space includes infeasible solutions. In such cases, penalty terms are often added to evaluate the degree of infeasibility [8]. It is also effective to dynamically change the evaluation function during the search, like in the noising method [6] and the search space smoothing method [13]. Another technique consists in developing new more informative evaluation functions which may not be directly related to the objective function such in [19].

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 the Formula 6, where d_k refers to the number of absolute differences with value k between adjacent vertices of the graph.

$$LA(G, \varphi) = \sum_{k=1}^{n-1} k d_k \tag{6}$$

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 Φ evaluation function presented in [31], 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 $\lfloor \Phi \rfloor$ results into the same integer value produced by Eq. 1 and 6.

The main idea of Φ 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 β 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 Eq. 7.

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

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 Φ 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.

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

[2] $\beta(G, \varphi) = Max\{|\varphi(u) - \varphi(v)| : (u, v) \in E\}$

The choice of Φ 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 [31] and confirmed by those described in Subsection 5.2. But also because Φ 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 Subsection 3.2, then only the $|A(u)| + |A(v)|$ absolute differences that change should be recomputed to update the value of Φ. It is faster than the $O(|E|)$ instructions originally required.

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 $\mathscr{A}$. 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 [25]. 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 position 2 through n, select one of the unplaced vertices for placement in the current position by using the frontal increase minimization strategy. It consists in selecting for placement i a vertex that is adjacent to the fewest vertices in $U_i - F_i$, where $F_i = \{u \in U_i : v \in P_i \text{ and } (u, v) \in E\}$ denote the *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: $tl_i(v) = |\{(u, v) \in E : u \in P_i\}|$ and $tr_i(v) = d(v) - tl_i(v)$, where $d(v)$ denotes the degree of the vertex v. Both measures are used to define a new selection factor $sf_i(v) = tr_i(v) - tl_i(v)$, which is used at the two-step general strategy described above as follows: For each placement i in step 2, select $v \in F_i$ 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-Φ using the method proposed by Varanelli and Cohoon in [35]. Their work is based on large-scale numerical studies, conducted by different authors [1, 14, 26, 36], 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 σ_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:

$$C_k \approx C_\infty - (\sigma_\infty^2 / T_k) \tag{9}$$

$$\sigma_k \approx \sigma_\infty \ , \tag{10}$$

where C_∞ and σ_∞, 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:

$$T_k(i) \approx \frac{\sigma_\infty^2}{C_\infty - c(i) - \gamma_\infty \sigma_\infty} \tag{11}$$

In equation 11, the parameter γ_∞ 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:

$$P[C_\infty - \gamma_\infty \sigma_\infty < X < C_\infty + \gamma_\infty \sigma_\infty] \approx 1 - |r|^{-1} \tag{12}$$

The reader is referred to [35] 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_∞ and the standard deviation of cost over the solution space σ_∞. Next, a heuristic solution with cost $c(i)$ is obtained using the method described in Subsection 3.4. Then, the offset γ_∞ 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_∞, σ_∞, $c(i)$ and γ_∞ 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-Φ 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 [2, 14, 17, 35]. In the TSSA-Φ implementation we preferred the statistical cooling schedule proposed in [2] because it has demonstrated to be very effective [1, 29] in several combinatorial optimization problems.

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

$$T_k = T_{k-1}\left(1 + \frac{\ln(1+\delta)T_{k-1}}{3\sigma_{T_{k-1}}}\right)^{-1} \tag{13}$$

where $\sigma_{T_{k-1}}$ is the standard deviation of the evaluation function values at the current temperature and δ is called the *distance parameter*. Small δ 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 $\overline{SC}_k < \epsilon$ is met. We call ϵ 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 and some of its important components. For these experiments, the algorithms were coded in C and compiled with *gcc* using the optimization flag -*O*3. 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 aver-

aged 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_i computed with the procedure described in Subsection 3.5.
b) Cooling schedule distance parameter $\delta = 0.10$.
c) Maximum neighboring solutions per temperature r:

$$r = \begin{cases} 5.0\text{E+}05 & \text{if } 1 < |E| \leq 500 \\ 2.0\text{E+}06 & \text{if } 501 < |E| \leq 50000 \\ 3.5\text{E+}06 & \text{if } 50001 < |E| \leq 1.1\text{E+}06 \\ 7.0\text{E+}06 & \text{if } |E| > 1.1\text{E+}06 \end{cases} \tag{14}$$

d) Stop factor ϵ =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 [28] and used later in [4, 22, 30, 31, 33]. 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 9 very large 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 [22] and later by Safro *et al.* in [33]. Their number of vertices is between 78136 and 1017253.

To assess the performance of our TSSA-Φ, 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-Φ algorithm with respect to the state-of-the-art heuristics, using the test-suite proposed in [28]. In the second part TSSA-Φ is compared against two previously reported best algorithms over 9 very large graphs [22].

[3] `www.lsi.upc.es/~jpetit/MinLA/Experiments`
[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-Φ and the following state-of-the-art heuristics: SS+SA [27, 28], BDT+SA [4], MS [22], AMG [33] and MA [31]; over 21 benchmarks proposed in [28].

						TSSA-Φ				
Graph	$\|V\|$	$\|E\|$	IFIM	C	Reference	C	$Avg.$	$Dev.$	T	Δ_C
randomA1	1000	4974	972583	867214	[31]	866968	866975.4	8.2	86.5	-246
randomA2	1000	24738	6724470	6528780	[28]	6522206	6522221.6	43.0	181.0	-6574
randomA3	1000	49820	14493215	14238712	[31]	14194583	14194585.5	2.6	279.1	-44129
randomA4	1000	8177	1845977	1718746	[31]	1717176	1717179.6	3.1	90.0	-1570
randomG4	1000	8173	280961	140211	[31, 33]	140596	140597.0	1.1	76.5	385
bintree10	1023	1022	71517	3696	[33]	3696	3697.1	1.6	38.8	0
hc10	1024	5120	523776	523776	[4, 22, 31, 33]	523776	523776.0	0.0	1.2	0
mesh33x33	1089	2112	44430	31729	[22, 33]	31856	31904.2	62.5	89.9	127
3elt	4720	13722	481815	357329	[31, 33]	359151	359176.0	26.4	1030.8	1822
airfoil1	4253	12289	384013	272931	[33]	276381	276866.5	511.8	982.1	3450
whitaker3	9800	28989	1231912	1144476	[33]	1143645	1145304.7	1145.3	3330.1	-831
c1y	828	1749	70922	62262	[31, 33]	62230	62234.4	3.8	32.8	-32
c2y	980	2102	90347	78822	[33]	78757	78810.8	159.3	46.7	-65
c3y	1327	2844	151622	123376	[31]	123145	123151.1	4.8	93.3	-231
c4y	1366	2915	131106	115051	[31]	114936	114971.6	93.8	88.1	-115
c5y	1202	2557	118541	96878	[31]	96850	96877.2	62.4	69.2	-28
gd95c	62	144	525	506	[4, 31, 33]	506	506.1	0.3	2.1	0
gd96a	1096	1676	146407	95242	[31]	95263	95277.9	19.6	61.0	21
gd96b	111	193	1497	1416	[31, 33]	1416	1417.8	2.9	2.7	0
gd96c	65	125	537	519	[4, 22, 28, 31, 33]	519	519.1	0.3	2.8	0
gd96d	180	228	2937	2391	[31, 33]	2391	2394.2	6.7	5.7	0
Average				1.257E+06		1.255E+06				-2286.5

4.2 Comparing TSSA-Φ with the state-of-the-art algorithms

The purpose of this experiment is to carry out a performance comparison of our TSSA-Φ algorithm, presented in Section 3, with respect to the following well-known heuristics: SS+SA [27, 28], BDT+SA [4], MS [22], AMG [33] and MA [31].

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-Φ algorithm, which were generated with the IFIM [25] heuristic detailed in Subsection 3.4. IFIM is very fast and it is able to compute a good quality solution for the largest graph in this test-suit (*whitaker3*) in 0.18 seconds. 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 ($Avg.$), its standard deviation ($Dev.$) and the average CPU time (T) in seconds 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 (Δ_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 6 instances. In several benchmarks the improvement achieved leads to a significant decrease of the cost (Δ_C up to -44129). For the other 5 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 [6] .

This information allows us to observe that the running time of SS+SA for the largest graph in this test-suit (*whitaker3*), was over 11 hours using a computer (K6 III at 600MHz) 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 hours using a machine (Pentium III at 600 MHz) 3 times slower; MS, 127.79 seconds in a CPU (Pentium III at 700 MHz), which is about 2.6 times slower; MA, 15299.72 seconds 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 [33] (Pentium IV at 1.7 GHz), which is approximatively 0.2% slower than ours. Then, it was executed on the graph *whitaker3* resulting in a CPU time of 280.28 seconds.

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 [22] and AMG [33]; but our algorithm is much faster than the previous proposed evolutionary approach MA [31]. 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.

[6] `www.tomshardware.fr/cat.php?c=21, histoire et hit parade des CPU.`

Table 2
Performance comparison of TSSA-Φ over a set of 9 very large benchmarks proposed in [22].

(a) TSSA-Φ is stopped when it begins to improve the previous best-known solution.

			MS		AMG		TSSA-Φ			
Graph	$\|V\|$	$\|E\|$	C	T	C	T	C	T	Δ_C	Δ_T
tooth	78136	452591	255465042	10.5	227639682	27.0	227634482	52.7	-5.20E+03	0.95
ocean	143437	409593	141732687	13.5	118882522	72.0	118880582	147.8	-1.94E+03	1.05
mrngA	257000	505048	348448986	23.5	305560971	90.0	305503135	290.8	-5.78E+04	2.23
rotor	99617	662431	247583742	16.5	221832991	42.0	221832985	573.1	-6.00E+00	12.65
598	110971	741934	340886008	19.0	281033967	57.0	281032048	392.3	-1.92E+03	5.88
144	144649	1074393	772846779	28.5	745701842	84.0	745682093	407.1	-1.97E+04	3.85
m14b	214765	1679018	1004606217	40.0	857743008	130.0	-	-	-	-
mrngB	1017253	2015714	3558254373	98.0	3254023540	520.0	3253970040	1529.6	-5.35E+04	1.94
auto	448695	3314611	4501150138	100.0	3871472605	340.0	-	-	-	-
Average									-2.00E+04	4.08

(b) TSSA-Φ is stopped when the evaluation function mean value shows only very small changes (see Subsection 3.7).

	IFIM		TSSA-Φ					
Graph	C	T	C	*Avg.*	*Dev.*	T	Δ_C	Δ_T
tooth	253104451	0.09	214678297	214884320.5	326043.8	361.6	-1.30E+07	12.39
ocean	210694459	0.10	114722301	114908560.6	219666.8	553.5	-4.16E+06	6.69
mrngA	697873953	0.19	294042454	294819720.3	798840.4	639.4	-1.15E+07	6.10
rotor	286752751	0.14	221696343	222045040.2	496501.4	952.4	-1.37E+05	21.68
598	351481792	0.20	280075154	280567592.1	721882.8	743.6	-9.59E+05	12.05
144	915001621	0.26	736442970	736844481.8	449505.6	770.3	-9.26E+06	8.17
m14b	1013429382	0.39	859059510	859676014.1	854704.9	4183.3	1.32E+06	31.18
mrngB	8103013010	0.82	3178145258	3178689877.2	806108.0	3254.4	-7.59E+07	5.26
auto	5750501188	0.95	4049967166	4050372915.5	775540.9	4842.1	1.78E+08	13.24
Average							-1.64E+07	12.97

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 [22] and AMG [33]. In this section we present a comparison of the results attained by the TSSA-Φ algorithm, over this test-suite [22], with respect to those obtained by the MS and AMG heuristics.

The results presented in Subsection 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 in 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 Subsection 3.7.

The data obtained in these comparisons is compiled in Table 2(a) and 2(b), 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 (Δ_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 (Δ_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 obtained it. The data in columns 8 to 11 from Table 2(a) was 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 2(b) for informative purpose.

Analyzing the data presented in Table 2(a) and 2(b) 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 7 out of 9 instances, achieving an average improvement of −2.00E+04 (see column Δ_C in Table 2(a)). For certain benchmarks, like the graph *mrngA*, an important reduction in cost is accomplished by our algorithm (Δ_C = −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 Δ_C =1.78E+08. We belive 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 2(b)) that TSSA-Φ is able to continue improving the solution quality when the stop condition described in Subsection 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 Δ_C = −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 [22, 33].

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

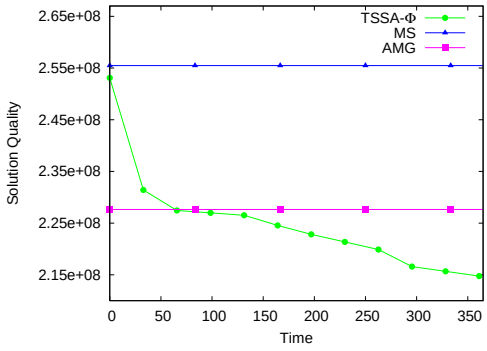


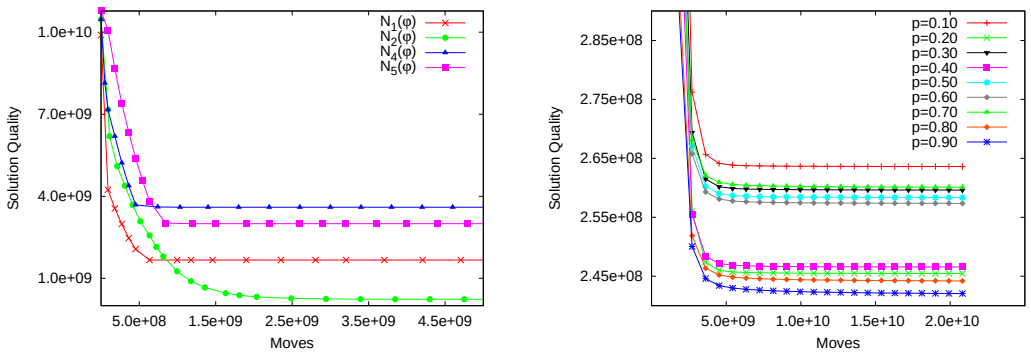
Figure 2. Convergence process of the average solution quality obtained by TSSA-Φ with respect to the consumed CPU time (in minutes) over the *tooth* graph. The stop condition described in Subsection 3.7 was used.

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 223988122.2 expending only 5.34 longer time than the AMG heuristic (i.e. 171.10 minutes), which represents an improvement of $\Delta_C = -3.65\text{E}{+}06$. The maximal amelioration in cost (214884320.5 vs. 227639682 for AMG) is obtained using 361.56 minutes.

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 Subsection 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.



(a) **(b)**

Figure 3. (a) Performance comparison of four neighborhood functions using the OSSA algorithm. (b) Performance comparison with 9 values of p for the $N_3(\varphi, x)$ neighborhood function.

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 Subsection 3.2, but also two other neighborhood relations defined in Eq. 15 and 16, where $swap(\varphi, u, v, w)$ is the result from applying the operations $swap(\varphi, u, v)$ and $swap(\varphi, v, w)$ successively.

$$N_4(\varphi) = \{\varphi' \in \mathscr{A} : swap(\varphi, u, v) = \varphi',\ (u, v) \in V,\ v \in A(u)\} \quad (15)$$

$$N_5(\varphi) = \{\varphi' \in \mathscr{A} : swap(\varphi, u, v, w) = \varphi',\ (u, v, w) \in V,\ u \neq v \neq w\} \quad (16)$$

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 observed that the best performance is attained by OSSA when the $N_2(\varphi)$ neighborhood function is used ($LA = 235301805.89$). 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 $N_3(\varphi, x)$, was presented in Formula 5, where p represents the probability to apply $N_1(\varphi)$ (the rest of the time $N_2(\varphi)$ is employed).

In order to find the most suitable value for the probability p, used in $N_3(\varphi, x)$, we proceeded as follows: For each one of the 9 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 $N_3(\varphi, x)$ neighborhood function ($LA = 225943530.37$). It is important to remark that this probability allows our combined $N_3(\varphi, x)$ neighborhood function to obtain even better results than $N_2(\varphi)$.

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 Φ evaluation function (see Subsection 3.3) and provide more insights into its real working.

First, a study of the Φ 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 $f(\varphi)$ starts from the initial solution $\varphi \in \mathscr{A}$ and repeats replacing φ with the best solution in its neighborhood $N_2(\varphi)$ until no better arrangement is found. Lets call SD-LA and SD-Φ the algorithm depending on which evaluation function it uses.

The results of this comparative study were presented in detail in [31]. From these results one observes that the SD algorithm that employs Φ, consistently has better results than the algorithm that uses LA in all the tested instances. The advantage of using Φ as evaluation function is well summarized in Fig. 4, where the behavior of the studied evaluation functions is presented over the $randomA1$ 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-Φ, basically because LA can not 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 axis) with respect

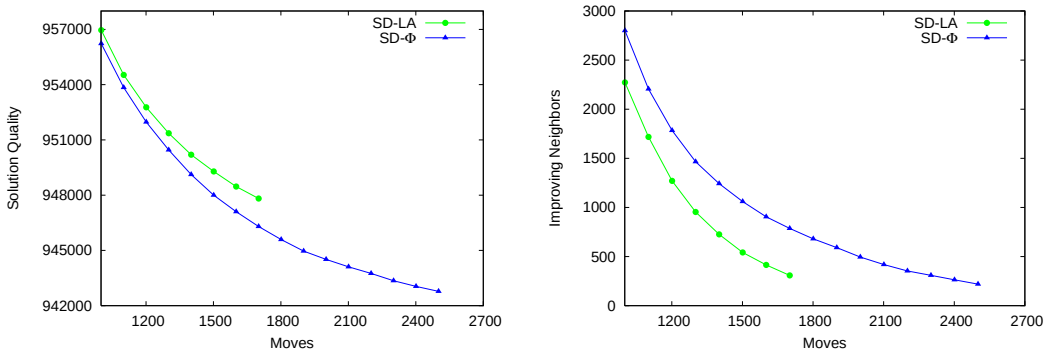


Figure 4. Graphs representing the behavior of the compared evaluation functions over the *randomA*1 instance using a SD algorithm. (a) Average solution quality, (b) Average improving neighbors.

to the number of moves is depicted. From this figure we can observe that SD-Φ produces better results because it is capable of identifying more improving neighbors and to distinguish those that orient better the search process. Additionally, these excellent results can be obtained without incrementing the computing time.

After having studied the characteristics of Φ 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-Φ 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-Φ 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 *randomA*1 instance. The ordinate represents the solution quality, while the abscissa indicates the number of moves. From this graph it can be seen that OSSA-Φ is more effective in searching better solutions than OSSA-LA at every instant of the search process. Indeed, it is possible because Φ is able to overcome the disadvantages presented by the classic evaluation function (see Subsection 3.3).

5.3 Influence of the cooling schedules

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 [21] and statistical [2]. Then, we have implemented them within the OSSA algorithm described at the begin of Section 5. We named them OSSA-*Geometric*

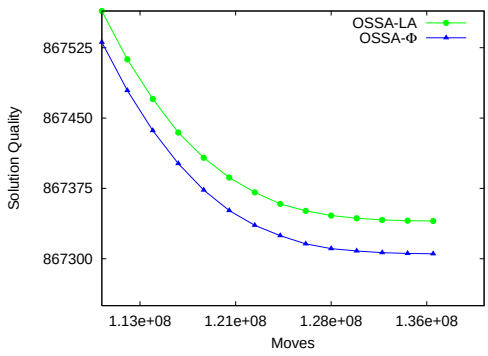


Figure 5. Performance comparison between the LA and Φ evaluation functions over the $randomA1$ instance using the OSSA algorithm.

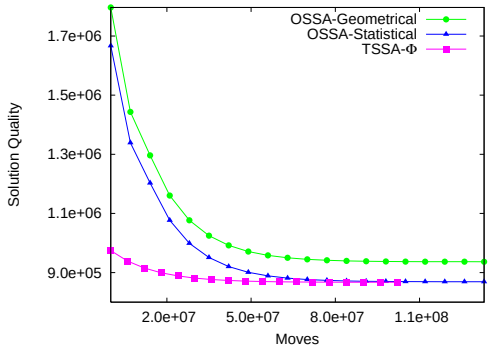


Figure 6. Performance comparison among three different cooling schedules over the $randomA1$ instance.

or OSSA-*Statistical* to distinguish which cooling schedule is employed.

For both algorithms the method described in [2] 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: $T_k = T_{k-1} * 0.96$. 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 Subsection 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-Φ presented in 3 for comparative purposes. Remember that TSSA-Φ uses also the statistical cooling schedule, but it starts from a good quality solution and at a suitable temperature determi-

nated by the method proposed in [35].

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-Φ algorithm with respect to the other two compared algorithms. The experiments carried out, over the whole test-suit, show that TSSA-Φ 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-Φ), 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 [28] and 9 very large instances from [22]. 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 "Contrat 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

[1] E. H. Aarts and J. H. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Sons, 1989.

[2] E. H. Aarts and P. J. V. Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Philips Journal of Research*, 40:193–226, 1985.

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

[4] R. Bar-Yehuda, G. Even, J. Feldman, and J. Naor. Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. *Journal of Graph Algorithms and Applications*, 5(4):1–27, 2001.

[5] K. D. Boese and A. B. Kahng. Best-so-far vs. where-you-are: Implications for optimal finite-time annealing. *Systems and Control Letters*, 22(1):71–78, 1994.

[6] I. Charon and O. Hudry. The noising method: A new method for combinatorial optimization. *Operations Research Letters*, 14(3):133–137, 1993.

[7] J. Diaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[8] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[10] J. Greene and K. J. Supowit. Simulated annealing without rejected moves. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(1):221–228, 1988.

[11] L. K. Grover. A new simulated annealing algorithm for standard cell placement. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 378–380, Santa Clara, CA, USA, 1986. IEEE Press.

[12] L. K. Grover. Standard cell placement using simulated sintering. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 56–59, Miami Beach, FL, USA, 1987. IEEE Press.

[13] J. Gu and X. Huang. Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man and Cybernetics*, 24(5):728–735, 1994.

[14] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–329, 1988.

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

[16] R. V. Hogg and A. T. Craig. *Introduction to Mathematical Statistics*. Macmillan, New York, 1970.

[17] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli. An efficient general cooling schedule for simulated annealing. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 381–384, Santa Clara, CA, USA, 1986. IEEE Press.

[18] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[19] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

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

[21] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[22] Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. *Lecture Notes in Computer Science*, 2573:293–306, 2002.

[23] Y.-L. Lai and K. Williams. A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *Graph Theory*, 31:75–94, 1999.

[24] J. Lam and J.-M. Delosme. Performance of a new annealing schedule. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 306–311, Atlantic City, NJ, USA, 1988. IEEE Press.

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

[26] R. Otten and L. V. Ginneken. Stop criterion in simulated annealing. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 549–552, Rye Brook, NY, USA, 1988. IEEE Press.

[27] J. Petit. Combining spectral sequencing and parallel simulated annealing for the MinLA problem. *Parallel Processing Letters*, 13(1):71–91, 2003.

[28] J. Petit. Experiments on the minimum linear arrangement problem. *The ACM Journal of Experimental Algorithmics*, 8, 2003.

[29] E. Poupaert and Y. Deville. Simulated annealing with estimated temperature. *AI Communications*, 13(1):19–26, 2000.

[30] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez. Memetic algorithms for the MinLA problem. *Lecture Notes in Computer Science*, 3871:73–84, 2006.

[31] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez. A refined evaluation function for the MinLA problem. *Lecture Notes in Computer Science*, 4293:392–403, 2006.

[32] J. Rose, W. Klebsch, and J. Wolf. Temperature measurement and equilibrium dynamics of simulated annealing placements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(3):253–259, 1990.

[33] I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

[34] P. N. Strenski and S. Kirkpatrick. Analysis of finite length annealing schedules. *Algorithmica*, 6(1):346–366, 1991.

[35] J. M. Varanelli and J. P. Cohoon. A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems. *Computers & Operations Research*, 26(5):481–503, 1999.

[36] S. R. White. Concepts of scale in simulated annealing. In *Proceedings of the IEEE International Conference on Computer Design*, pages 646–651, Port Chester, NY, USA, 1984. IEEE Press.