ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20130905130612

BOOSTING THE PERFORMANCE OF METAHEURISTICS FOR THE MINLA PROBLEM USING A MORE DISCRIMINATING EVALUATION FUNCTION

Eduardo Rodriguez-Tello, Jin-Kao Hao, Hillel Romero-Monsivais

Original scientific paper
This paper investigates the role of evaluation function used by metaheuristics for solving combinatorial optimization problems. Evaluation function (EF) is a key component of any metaheuristic algorithm and its design directly influences the performance of such an algorithm. However, the design of more discriminating EFs is somewhat overlooked in the literature. We present in this work the first in-depth analysis of the conventional EF for the Minimum Linear Arrangement (MinLA) problem. The results from this study highlighted its potential drawbacks and led to useful insight and information which guided us to design a new more discerning EF. Its practical usefulness was assessed within three different algorithms: a parameter-free Steepest Descent, an Iterated Local Search and a Tabu Search. The analysis of the data produced by these comparisons showed that the performance of the three adopted approaches could be boosted by using the proposed more discriminating EF.

Keywords: *combinatorial optimization, evaluation function, linear arrangement problem, metaheuristics*

Poboljšanje učinaka metaheuristike kod MinLA problema primjenom kritičnije funkcije evaluacije

Original scientific paper
U radu se ispituje uloga funkcije evaluacije u metaheuristici kod rješavanja kombinatornih problema optimizacije. Evaluacijska funkcija (EF) je ključna sastavnica svakog metaheurističkog algoritma i njezin dizajn direktno utječe na performansu takvog algoritma. Međutim, u literaturi je dizajn kritičnijih EF-a donekle zanemaren. U ovom radu dajemo prvu temeljnu analizu standardne EF za problem Minimum Linear Arrangement (MinLA). Dobiveni rezultati su ukazali na moguće nedostatke i dali koristan uvid i informacije potrebne za dizajniranje kritičnije EF. Njezina se praktična korisnost procijenila u tri različita algoritma: parameter-free Steepest Descent, Iterated Local Search i Tabu Search. Analiza dobivenih podataka pokazala je da bi se performansa ta tri primijenjena pristupa mogla poboljšati primjenom predloženih kritičnijih EF.

Ključne riječi: *kombinatorna optimizacija, funkcija evaluacije, problem linearnog uređenja, metaheuristika*

1 Introduction

In the last two decades, metaheuristics [1] such as Genetic Algorithms, Simulated Annealing and Tabu Search have become very popular as a class of valuable optimization methods for tackling hard combinatorial optimization problems. Successful applications with these methods are continually reported both in traditional and in emerging fields. Metaheuristics are today recognized as an indispensable part of the arsenal for difficult optimization.

The success (or failure) of a metaheuristic algorithm depends heavily on a set of key components that must be designed with care. Neighborhood relation and evaluation function are two prominent examples for Stochastic Local Search methods [2]. The neighborhood relation defines the subspace of the search problem to be explored by the method. For a given problem, the definition of the neighborhood should structure the search space such that it helps the search process to find its ways to good solutions. The importance of neighborhood is also evidenced by several methods focusing on neighborhood relations like Variable Neighborhood Search (VNS) [3], Neighborhood Portfolio Search [4] and Progressive Neighborhood Search [5].

The evaluation function assesses the quality of a candidate solution with respect to the optimization objective and orients the search method to "navigate" through the search space. A good evaluation function is expected to be able to distinguish among solutions and thus to effectively guide the search method to make the most appropriate choice at each of its iterations. Together, neighborhood and evaluation function define the so-called landscape of the search problem [2, 6, 7] and impact thus greatly the efficiency of the search algorithm.

One common practice in designing metaheuristic algorithms is to directly use the initial objective function of the optimization problem as the evaluation function. However, such a function may not be sufficient to effectively guide the search process. Consider for instance the more discriminating evaluation functions reviewed in Section 5.

In this paper, we focus on the issue of evaluation function and offer an in-depth investigation on the design of a more discriminating evaluation function which considers additional semantic information of the optimization problem. For this purpose, we use the well-known Minimum Linear Arrangement problem (MinLA) as our case study.

The main contributions of this work can be summarized as follows: a) An in-depth analysis of some mathematical properties of the classical evaluation function for MinLA, called L_A, and a new more discriminating evaluation function, named Φ; b) An extensive experimental comparison between these evaluation functions within a Steepest Descent (SD) algorithm over a full test-suite composed of the 21 well-known benchmarks previously used in many studies [8, 9, 10, 11, 12]; c) A study comparing the number and distribution of neutral and improving neighboring solutions induced by both evaluation functions; d) An analysis of the interaction among five different neighborhood relations and the studied evaluation functions; e) An assessment of the practical usefulness of Φ within two different metaheuristic algorithms, Iterated Local Search and Tabu Search; f) A rigorous statistical analysis of all the experimental results. The main

objective of this investigation is to show that designing more discriminating evaluation functions may be a highly valuable approach for boosting the performance of metaheuristic algorithms which deserves more attention.

The remainder of this article is organized as follows. Section 2 formally introduces the MinLA problem, analyzes some characteristics of the basic evaluation function L_A and shows how these results were used in [13] to propose the more informative Φ evaluation function. Section 3 shows a first assessment of the usefulness of Φ with respect to the conventional L_A function, with the help of a parameter-free Steepest Descent (SD) algorithm. Additional evidence about the usefulness of the proposed evaluation function within both an Iterated Local Search and a Tabu Search algorithm is given in Section 4. Section 5 provides a review of some relevant studies related to the issue of more discriminating evaluation functions. Finally, Section 6 summarizes the contributions of this paper and highlights the importance of designing alternative evaluation functions for metaheuristic algorithms.

2 The MinLA problem and evaluation functions

Let $P = (S, f)$ be a given combinatorial optimization problem where S is the search space composed of a set of candidate solutions and f the objective or cost function of problem P. For the purpose of this paper, it is important to distinguish f from the notion of evaluation function g (also called fitness function for genetic-like algorithms) which is a component of a metaheuristic algorithm. In many cases, g can take the form of f and many examples can be found using such an approach. However, g can also be defined by any other function in order to include in it additional and useful information. This second approach is the topic of this paper. To show how this would be possible, we focus on the study of the *Minimum Linear Arrangement* problem (MinLA).

MinLA was first stated by Harper [14] whose initial aim was to design error-correcting codes with minimal average absolute errors on certain classes of graphs. Later, in the 1970's 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 [15]. MinLA has also been applied as an over-simplified model of some nervous activity in the cortex [16]. The MinLA problem also has other practical applications, particularly in the following areas: bioinformatics [17], single machine job scheduling [18], graph drawing [19], software diagram layout [20], to mention only some of them.

2.1 The L_A function

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

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

Then the MinLA problem consists in finding an arrangement (labeling) φ^* for a given G so that *the total edge length* $L_A(G, \varphi)$ is minimized:

$$L_A(G, \varphi^*) = \min\{L_A(G, \varphi) : \varphi \in \mathcal{L}\}, \quad (2)$$

where $\mathcal{L}$ represents the set of all the possible labelings. It is simple to observe that the set $\mathcal{L}$ consists of $n!$ possible linear arrangements for a graph of order n.[1]

There exist polynomial time exact algorithms for some special cases of MinLA such as trees, rooted trees, hypercubes, meshes, outerplanar graphs, and others (see [21] 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 [22]. 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 Binary Decomposition Tree heuristic [9], the Multi-Scale algorithm [10] and the Algebraic Multi-Grid scheme [23]; b) metaheuristics such as Simulated Annealing [8, 12] and Memetic Algorithms [11, 13]; and c) some combinations of these methods.

2.2 Analyzing the L_A evaluation function

It is important to remark that most of the algorithms for the MinLA problem mentioned in the previous section, evaluate the quality of a solution (linear arrangement) as the change in the objective function $L_A(G, \varphi)$ (let us call it only L_A for simplicity). This section presents a detailed analysis of certain characteristics of L_A and highlights the potential drawbacks of L_A when it is directly used as an evaluation function. This analysis has led to useful insight and information which guided us to design a more discriminating evaluation function introduced in [13].

Before beginning this analysis, we introduce two important definitions used in our study of labeled graphs. Let φ be a labeling for a graph $G(V, E)$ of order n. The *absolute difference* between the labels of two adjacent vertices u, v is defined as follows: $|\varphi(u) - \varphi(v)| = k$ for $(u, v) \in E$ and $0 \leq k \leq n - 1$.

The *appearing frequency* d_k of those absolute differences with value k produced by φ is given by Eq. (3).

$$d_k = \sum_{(u,v) \in E} l_{uv}, \quad (3)$$

where $(u, v) \in E$ equals 1 if $|\varphi(u) - \varphi(v)| = k$, and 0 otherwise.

Let $G(V, E)$ be a finite undirected graph of order n and φ a labeling. Observe that graph G could potentially have $n(n-1)/2$ non-reflexive edges. Given that a

[1] Notice that each of the $n!$ linear arrangements may be reversed to get the same cost.

particular edge can be present or absent in it, the total number of possible simple (without loops) labeled graphs is expressed in Eq. (4), where $\mathcal{G}_s$ represents the set containing those graphs.

$$|\mathcal{G}_s| = 2\frac{n(n-1)}{2} = 2\binom{n}{2} \tag{4}$$

In the following analysis, we do not consider the graphs with loops because the contribution of those non-reflexive edges to the total edge length is null. Nevertheless, we take into consideration all the possible values taken by the function LA, even when $\text{LA} = 0$.

Suppose a labeled graph with *n* vertices. In this graph the maximum number of absolute differences is distributed as follows: 1 with value $(n-1)$, 2 with value $(n-2)$, and in general k absolute differences with value $(n-k)$ for all k in $[1, n-1]$. Then, the LA function can be expressed in terms of the graph's appearing frequencies d_k using Eq. (5).

$$\text{LA}(G, \varphi) = \sum_{k=1}^{n-1} k d_k \tag{5}$$

It is clear then, that LA can take values from 0 (an empty graph or composed only by reflexive edges) to the value given by Eq. (6).

$$\sum_{k=1}^{n-1} k(n-k) = \frac{n(n^2-1)}{6} \tag{6}$$

Let $\mathcal{R}_{\text{LA}} \subseteq \mathcal{G}_s \times \mathcal{G}_s$ be an *equivalence relation* [24] over the set of all the simple labeled graphs and $x, y \in \mathcal{G}_s$then x is related to y ($x\ \mathcal{R}_{\text{LA}}\ y$) if and only if x and y have a set of appearing frequencies $D = \{d_1, d_2, \ldots, d_{n-1}\}$ which produce the same total edge length LA.[2] An *equivalence class*$[x] = \{y \in \mathcal{G}_s : x\ \mathcal{R}_{\text{LA}}\ y\} \subseteq \mathcal{G}_s$ then can be defined for each possible value of LA. Hence, the total number of equivalence classes ε_{LA}, underLA, in which the set $\mathcal{G}_s$ can be partitioned is:

$$\varepsilon_{\text{LA}} = 1 + \frac{n(n^2-1)}{6}$$

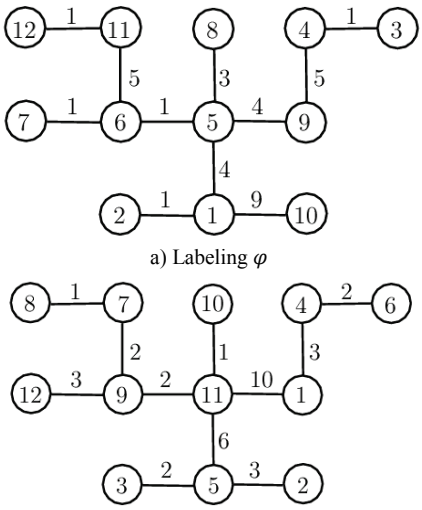
We observe that that there can exist different labeling resulting into the same set of appearing frequencies *D* whose exact number z can be computed using Eq. (7).

$$z(D) = \prod_{k=1}^{n-1} \binom{k}{d_{n-k}} \tag{7}$$

Notice also that it could exist p different sets of appearing frequencies resulting into the same value LA, i.e., $P = \{D_1, D_2, \ldots, D_p\}$. Thus, the cardinality $\omega_{\text{LA}}(i)$for the equivalence class under $\text{LA} = i$ can be computed using Eq. (8).

$$\omega_{\text{LA}}(i) = \sum_{j=1}^{p} z(D_j) \tag{8}$$

According to our observations, the evaluation function LA does not distinguish between absolute differences with a big value and those with a little value. For instance, for the LA function it is equivalent to have an absolute difference with value 25 rather than 25 absolute differences with value 1 (see Eq. (1)). Therefore, there is no possibility of making the distinction between the $\omega_{\text{LA}}(i, P)$simple labeled graphs which belong to the same equivalence class $\text{LA} = i$.



a) Labeling φ

b) Labeling φ'

Figure 1 Example of two linear arrangements with the same value LA = 35

For example, the LA function assigns the same cost $\text{LA} = 35$ for the two labelings represented in Fig. 1, for a simple graph of order 12. However, a deeper analysis of each appearing frequency $d_k$allows us to confirm that the labeling φ' in Fig. 1b is perhaps more interesting than that of Fig. 1a because it is easier to minimize the value of LA by reducing one absolute difference with value 10 ($d_{10} = 1$) in φ' rather than five absolute differences with value 1 ($d_1 = 5$) in φ. Indeed, if we reduced one absolute difference with value 10, LA could decrease by 10 units, whereas if we can eliminate five absolute differences with value 1 (which is more difficult), LA could only reduce its value by 5 units.

Based on this observation, we review in the next section an evaluation function for the MinLA problem which is more informative. Indeed, it analyzes individually each appearing frequency produced by a labeling in order to assess its quality.

2.3 The Φ evaluation function

The function Φ evaluates the quality of a labeling considering not only the total edge length (LA) of the arrangement, but also additional information induced by the appearing frequencies of the graph. Furthermore, it maintains the fact that $\lfloor \Phi \rfloor$ results into the same integer value produced by Eqs. (1) and (2).

[2] Observe that it could be the same graph or two different graphs.

For computing Φ, each appearing frequency d_k must make a different contribution to the cost of a labeling. This contribution is computed by using Eq. (9).

$$k + \frac{1}{\prod_{j=1}^{k}(n+j)} = k + \frac{n!}{(n+k)!} \tag{9}$$

Applying these contributions into Eq. (5) produces the following expression:

$$\sum_{k=1}^{n-1}\left(k + \frac{n!}{(n+k)!}\right)d_k\,, \tag{10}$$

and simplifying it, we obtain the Φ evaluation function whose first term is the LA function (see Eq. (5)), and the second term (a fractional value) is used to discriminate labelings having the same total edge length LA.

$$\Phi(G,\varphi) = \sum_{k=1}^{n-1} k d_k + \sum_{k=1}^{n-1}\frac{n!\,d_k}{(n+k)!} \tag{11}$$

As an example, let us consider the labeling φ for the graph of order $n = 12$ presented in Fig 1a. For this particular linear arrangement $L_A = 35$ and the appearing frequencies d_k are: d_1=5,d_3=1,d_4=2,d_5=2 and d_9=1. By substituting these values in Eq. (11) we get:

$$\begin{aligned}\Phi(G,\varphi) = 35 + 12!\,(&\frac{5}{6{,}2\mathrm{E}+09} + \frac{1}{1{,}3\mathrm{E}+12}\\ &+ \frac{2}{2{,}0\mathrm{E}+13} + \frac{2}{3{,}5\mathrm{E}+14}\\ &+ \frac{1}{5{,}1\mathrm{E}+19}) = 35{,}385\end{aligned} \tag{12}$$

On the other hand, if Φ is computed for the linear arrangement φ' depicted in Fig. 1b, we observe that the appearing frequencies d_k are: d_1=2,d_2=4,d_3=3,d_6=1 and d_{10}=1, which provide a smaller value:

$$\begin{aligned}\Phi(G,\varphi') = 35 + 12!\,(&\frac{2}{6{,}2\mathrm{E}+09} + \frac{4}{8{,}7\mathrm{E}+10}\\ &+ \frac{3}{1{,}3\mathrm{E}+12} + \frac{1}{6{,}4\mathrm{E}+15}\\ &+ \frac{1}{1{,}1\mathrm{E}+21}) = 35{,}177\end{aligned} \tag{13}$$

The main idea behind Φ is to penalize the absolute differences having small values, and to favor those with values near to the bandwidth β of the graph.[3] This can be clearly observed in Eq. (12), where the Φ function penalizes the absolute differences with value 1 (d_1=5) more than those with value 3 (d_3=1) by multiplying them by a factor of $12!/(6{,}2\mathrm{E}+09)$ and $12!/(1{,}3\mathrm{E}+12)$, respectively. In this way, Φ will always assign a lower cost for a labeling comprising more absolute differences with big value like that depicted in Fig. 1b, because intuitively this labeling has a stronger probability to be further improved.

Indeed, remember that in a simple labeled graph of order n, generally there is a maximum number $(n-k)$ of absolute differences with value $k (k \in [1, n-1])$. Consequently, a larger value of an absolute difference k means a weaker appearing frequency d_k. For example, in a simple labeled graph of order $n = 50$ there exist at most 49 absolute differences with value 1, whereas there is at most "one" absolute difference with value 49. It is thus simpler to reduce the total edge length for a given labeling by modifying the labels at the ends of only one edge, than changing the labels of all the vertices joined by the 49 edges which produce absolute differences with value 1. Based on this observation, we have conceived the Φ function in such a way that it can give advantage to the labelings which have absolute differences with big value.

2.4 Analyzing the Φ evaluation function

A formal analysis of certain characteristics of the Φ evaluation function is presented below.

Let $\mathcal{R}_\Phi \subseteq \mathcal{G}_s \times \mathcal{G}_s$ be an *equivalence relation* over the set of all the simple labeled graphs. Given two elements $x, y \in \mathcal{G}_s$, we say that x is related to y $(x\,\mathcal{R}_\Phi\,y)$ if and only if both have the same set of appearing frequencies $D = \{d_1, d_2, \ldots, d_{n-1}\}$(i.e., the same Φ value). An *equivalence class* $[x] = \{y \in \mathcal{G}_s : x\,\mathcal{R}_\Phi\,y\} \subseteq \mathcal{G}_s$ can be then defined for each possible value of Φ.

Remember that in a simple labeled graph of order n, there are in general k absolute differences with value $(n-k)$ for all k in $[1, n-1]$. Then, the appearing frequencies take values between 0 and $(n-k)$. Thus, the total number of equivalence classes ε_Φ, under Φ, in which the set $\mathcal{G}_s$ of all the simple labeled graphs of order n can be partitioned is:

$$\varepsilon_\Phi = \prod_{k=1}^{n-1}(n-k) = n! \tag{14}$$

Given that two labelings belong to the same equivalence class under Φ if they have the same set of appearing frequencies D, we can compute the cardinality $\omega_\Phi(i, D)$ for the equivalence class under $\Phi = i$ by using Eq. (15).

$$\omega_\Phi(i, D) = \prod_{k=1}^{n-1}\binom{k}{d_{n-k}} \tag{15}$$

The analysis of the equivalence classes produced by Φ, that we have just presented, allows us to draw some important conclusions. In particular, we observed the fact that Φ divides each equivalence class produced by the LA function in subsets (equivalence classes) of smaller size which gather labelings sharing the same set of appearing frequencies D. Thanks to this rational way of incrementing the number of equivalence classes, Φ makes it possible to distinguish linear arrangements having the same value LA. Moreover, Φis coherent with the MinLA problem objective which consists in minimizing LA: for

[3] Given a labeling function $\varphi$of graph G its bandwidth is: $\beta(G,\varphi) = \max\{|\varphi(u) - \varphi(v)| : (u,v) \in E\}$.

two labelings φ and φ' if $\Phi(\varphi) < \Phi(\varphi')$, then $L_A(\varphi) \leq L_A(\varphi')$(see Eqs. (5) and (11)).

2.5 Comparison of computational complexity between L_A and Φ

In order to compute the quality of a labeling $\varphi$by using the conventional L_A evaluation function, all the edges in the graph $G(V,E)$ must be analyzed (seeEq. (1)). As a result $O(|E|)$ instructions must be executed.

To efficiently compute Φ we could precalculate each term $k + (n!/(n+k)!)$ in Eq. (11) and store them in an array $M = \{m_{ij}\}_{n\times n}$.All this needs to execute $2|V|$ operations, i.e., $O(|V|)$. Then each time that we need to calculate the value of Φ the sum $\sum_{(u,v)\in E} m_{uv}$must be computed, which results into the same computational complexity as the one required to compute L_A. Additionally, Φpermits an incremental evaluation of neighboring solutions (see Section 3).[4] Indeed, suppose that the labels of two different vertices (u,v) are exchanged, then we should only recompute the $|A(u)| + |A(v)|$ absolute differences that change, where $|A(u)|$ and $|A(v)|$ represent the number of adjacent vertices to u and v, respectively. As it can be seen this is faster than the $O(|E|)$ operations originally required.

To complement the above analysis, we will present bellow experimental evidences confirming the advantage of the Φ evaluation function over the L_A function and show the usefulness of Φ for metaheuristic algorithms.

3 Comparing L_A and Φ evaluation functions using a Steepest Descent algorithm

This section has two main objectives. First, with a Steepest Descent algorithm and a set of benchmark instances, we compare the performances that can be achieved using L_A and Φ evaluation functions. Second, to explain the observed performance difference, we examine in detail the interaction between evaluation function and neighborhood relation and analyze the distribution of the improving neighbors produced by the evaluation function Φ in comparison with L_A.

3.1 Steepest Descent algorithm

The choice of the Steepest Descent (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 any bias. In addition, the move strategy adopted within the SD algorithm permits to study characteristics of the search space in which we are interested, such as the number and distribution of improving neighbors. The implemented SD algorithm has the following features.

Solution representation and Evaluation Function. For a graph G with n vertices, the search space $\mathcal{L}$ is composed of all $n!/2$ possible linear arrangements. In our SD algorithm, a linear arrangement φ is represented as a permutation of $\{1,2,\ldots,n\}$. More specifically, it is defined as an array l of n integers which is indexed by the vertices and whose i-*th* value $l[i]$ denotes the label assigned to the vertex i. The cost of an arrangement φ is evaluated by using either the L_A or Φ evaluation function.

Initial Solution. In this implementation the initial solution is randomly generated.

Neighborhood Function. Let $swap(\varphi,u,v)$ be a function permitting to exchange the labels of two vertices u and v from an arrangement φ. The *neighborhood*$\mathcal{N}_1(\varphi)$ of an arrangement φ can be then defined as follows:

$$\mathcal{N}_1(\varphi)=\{\varphi' \in \mathcal{L}: swap(\varphi,u,v)=\varphi', u,v \in V, u \neq v\} \quad (16)$$

This neighborhood has the advantage of being small and enabling an incremental cost evaluation of neighboring solutions.

SD Move Strategy and Stop Condition. The SD algorithm starts from the initial solution $\varphi \in \mathcal{L}$ and repeatedly replaces φ with the best solution in its neighborhood $\mathcal{N}_1(\varphi)$, ties are broken randomly. This process stops automatically when no better arrangement can be found within the neighborhood.

3.2 Computational experiments

This experiment aims at studying the characteristics of φ and at providing insight into its real working. That is why it does not only take into account the final solution quality obtained by the algorithms, but also their ability to efficiently explore the search space. To attain this objective, the SD algorithm presented in Section 3.1 was coded in C and named SD-L_A and SD-Φ depending on which evaluation function is used. This algorithm as well as all the other presented in this paper were compiled with *gcc* using the optimization flag *-O3*, and ran sequentially into a CPU Xeon at 2 GHz, 1 GB of RAM with Linux operating system.

The test-suite used in all our experiments is composed of the 21 well-known benchmarks originally proposed by Petit [8] and used later by many studies [9, 10, 11, 12]. These instances are divided into six different kinds of graphs having between 65 and 9800 vertices and are available at the following address: http://www.tamps.cinvestav.mx/~ertello/minla.php.

To assess the performance of the studied algorithms (SD-L_A and SD-Φ), comparative results are shown on these instances. The main criterion used for the comparison is the *best total edge length* found (smaller values are better). Computing time is also given for indicative purpose.

3.3 Comparison between SD-L_A and SD-Φ

The methodology used consistently throughout this experimentation is the following. First, 10 random arrangements were generated for each of the 21 selected instances. Then, each random arrangement was used as starting solution for 10 independent runs of the studied algorithms (SD-L_A and SD-Φ) over each selected instance. The average results achieved in these 100 executions are summarized in Tab. 1, where the first column indicates the name of the graph. Columns two to seven display the total iterations *I*, the final cost in terms

[4] Note that L_A can also be incrementally computed.

of total edge length C, and the total CPU time T in seconds for both SD-LA and SD-Φ. Column eight presents the average number of improving neighbors $\mathcal{N}_I$ found by SD-Φ, at the same iteration where the average number of improving neighbors provided by SD-LA equals zero, that is when SD-LA stops. Column Δ_C shows the percentage gain in terms of the average costs reached by SD-Φ compared to that produced by SD-LA, i.e. $100 * (1 - \textit{average cost of}$ SD-Φ / $\textit{average cost of}$ SD-LA).

Additionally, a statistical significance analysis was performed for this experiment. First, *D'Agostino-Pearson's omnibus* K^2 test was used to evaluate the normality of data distributions. For normally distributed data, either *ANOVA* or the *Welch's t* parametric tests were used depending on whether the variances across the samples were homogeneous (*homoskedasticity*) or not. This was investigated using the *Bartlett's* test. For non-normal data, the nonparametric *Kruskal-Wallis* test was adopted. A significance level of 0.05 has been considered. The resulting *p-value* is presented in Column ten of Tab. 1, where a bold typeface means that there exists a statistically significant increase in performance achieved by SD-Φ with regard to SD-LA.

The results presented in Tab. 1 show that the SD-Φ algorithm returns better results in 19 out of 21 instances (smaller total edge length) than those produced by SD-LA, which leads to an average improvement $\Delta_C = 3{,}33\%$ when Φ is used as evaluation function. Notice that the search with SD-Φ continues always longer time than with SD-LA (compare columns four and seven, labeled T), leading to better local optima. SD-LA stops prematurely due to its impossibility to distinguish neighboring solutions with the same cost (total edge length). It is important to remark that in average the CPU time needed for one iteration (T/I) of SD-LA and SD-Φ are quite comparable: 0,666 seconds for SD-LA and 0,656 seconds for SD-Φ.

The statistical analysis presented in the last column (*p-value*) of Tab. 1 confirms that there exists a statistically significant increase in performance achieved by SD-Φ with regard to SD-LA on 66,66 % of the studied instances (14 out of 21). Thus, we can conclude that in general SD-Φis more effective than SD-LA.

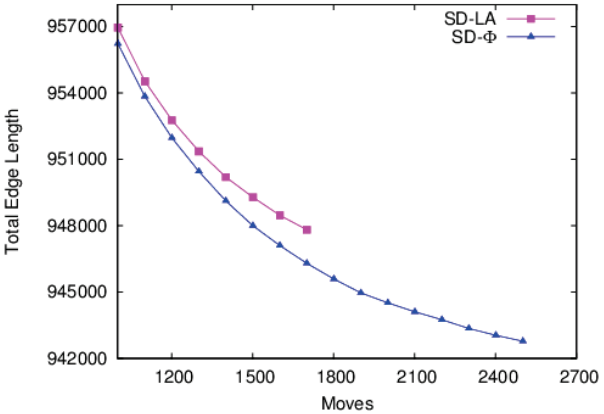
The dominance of Φ is better illustrated in Figs. 2a and 2b, where the behavior of the studied evaluation functions is presented on the *random* A1 instance (the study of other graphs provided similar results). In Fig. 2a the abscissa axis represents the number of moves, while the ordinate axis indicates the average total edge length. It can be observed that SD-Φ obtains always arrangements of shorter total edge length than SD-LA with the same number of moves. To supplement this observation, Fig. 2b (see also next section) depicts the evolution of the average number of improving neighbors (i.e., having a smaller cost than the current solution) with respect to the number of moves. One observes that at each iteration, Φ offers always more opportunities to select an improving neighbor than LA, favoring thus the SD algorithm to find better solutions.

3.4 Distribution of improving neighbors produced by LA and Φ

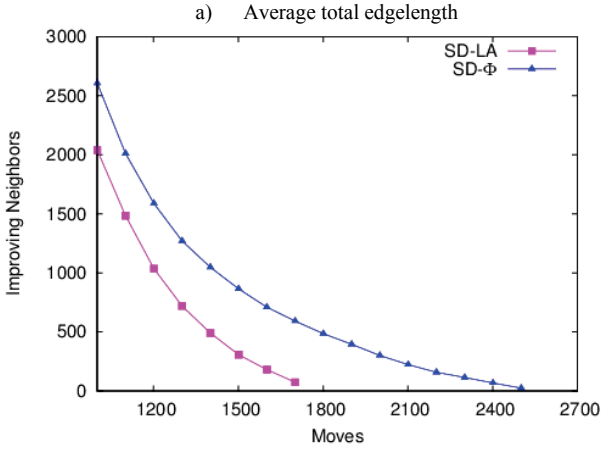
In order to understand and explain the results shown in Section 3.3, we present a second experiment for a deeper analysis. Here, we show the distribution of two types of neighbors: neutral neighbors (i.e., having the same cost as the current solution) and improving neighbors (i.e., having a smaller cost than the current solution) associated with LA and Φ. Indeed, the number of improving neighbors produced by an evaluation function (for a given neighborhood) is an interesting indicator of its capacity to allow the search algorithm to effectively explore the search space [25].

Table 1 Average results achieved in 100 executions of SD-LA and SD-Φ.

	SD-LA			SD-Φ				%	
Graph	I	C	T	I	C	T	$\mathcal{N}_I$	Δ_C	*p-value*
randomA1	2116,7	946 033,10	41,24	3135,2	941 677,30	56,68	234,7	0,46	**1,70E-04**
randomA2	2352,6	6 678 051,60	312,49	3115	6 673 334,30	461,45	289,2	0,07	6,00E-02
randomA3	2461,1	14 397 879,80	1190,35	2888,3	14 396 580,30	1351,2	327,4	0,01	9,40E-01
randomA4	2198	1 810 393,50	60	3041,1	1 807 026,10	83,87	250,8	0,19	**5,80E-03**
randomG4	2223,1	377 994,30	54,4	2366,2	381 002,30	57,22	255,8	-0,8	7,60E-01
bintree10	1234,6	51 716,80	16,24	1407,9	51 548,40	18,11	58,2	0,33	6,70E-01
hc10	1738,9	648 728,10	46,88	1959,9	650 515,30	35,8	361,4	-0,28	6,20E-01
mesh33x33	2995,4	130 751,30	45,67	8144,9	112 171,40	122,8	305,6	14,21	**1,10E-13**
3elt	27 119,40	2 630 144,60	7343,34	52 062,60	2 392 981,00	14 560,78	1150,8	9,02	**1,30E-08**
airfoil1	23 567,40	2 184 693,30	5622,94	45 910,10	1 958 983,10	10 386,57	862,7	10,33	**1,20E-14**
whitaker3	71 049,90	11 473 102,70	88 118,78	143 499,90	10 377 561,90	163 396,62	1788,1	9,55	**3,50E-14**
c1y	1737,9	122 959,20	29,79	2502,2	120 811,90	32,15	147,7	1,75	**1,20E-02**
c2y	2141,9	169 434,70	33,99	3079,9	167 127,50	51,97	153,4	1,36	1,90E-01
c3y	3335,1	282 818,50	186,3	5098,4	275 528,90	234,52	247,2	2,58	**1,00E-02**
c4y	3410,7	287 198,80	88,93	5421,9	280 551,40	155,79	249,1	2,31	6,30E-02
c5y	2811	233 286,10	61,07	4104,1	228 225,70	95,16	160,5	2,17	**1,10E-02**
gd95c	70,2	716,3	0,01	82,3	696,70	0,00	14,8	2,74	**1,40E-02**
gd96a	2216,7	148 158,30	36,94	3284,5	144 751,30	57,47	190,8	2,3	**7,60E-10**
gd96b	94,3	1857,3	0,03	109,1	1783,60	0,00	20,9	3,97	**1,40E-02**
gd96c	75,1	692,9	0,01	96,2	656,50	0,00	17,2	5,25	**1,40E-02**
gd96d	186,2	4166,3	0,11	257,3	4057,00	0,00	26,4	2,62	**1,10E-04**
Average	7387,4	2 027 656,10	4918,55	13 884,10	1 950 836,80	9102,77	338,7	3,34	14+



a) Average total edgelength



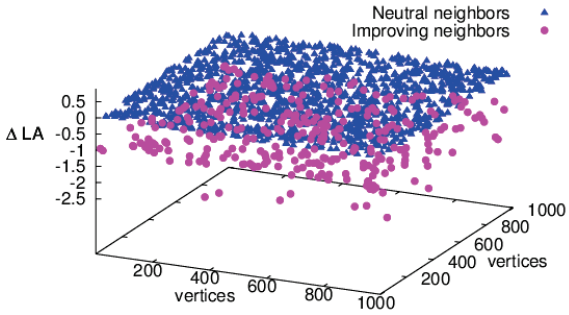
b) Average improving neighbors

Figure 2 Performance comparison between the algorithms SD-LA and SD-Φ over the *random* A1 instance

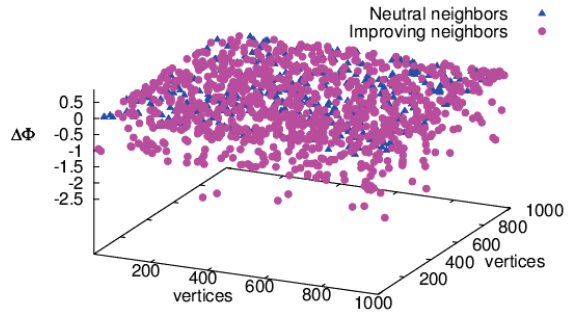
Figs. 3a and 3b show the distribution of all the neutral neighbors and improving neighbors produced, after 1500 iterations, by SD-LA and SD-Φ over the *random* A1 instance. The abscissa and ordinate axes indicate the vertices exchanged to produce $\mathcal{N}_1(\varphi)$, while the Z axis denotes the evaluation function cost produced. One observes that there is a great difference. SD employing the Φ evaluation function has many improving neighbors to ameliorate its current solution while SD-LA has much fewer favorable neighbors. This unfavorable characteristic of the LA evaluation function is dramatically accentuated as the search process progresses. For instance, observe Figs. 3c and 3d, which show also all the neutral and improving neighbors at the iteration where SD-LA stops. Notice how Φ is able to discriminate all those neutral neighbors that would be impossible to distinguish by using the classical evaluation function LA.

3.5 Interaction between the neighborhood relation and the evaluation function

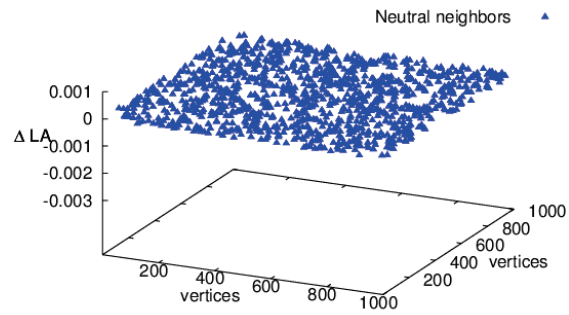
The neighborhood relation and the evaluation function together define the search landscape [6]. It is thus important to analyze the interaction between these two components. For this purpose, we consider the evaluation functions LA and Φ, the neighborhood relation $\mathcal{N}_1(\varphi)$ defined in Eq. (16), as well as four other neighborhood functions introduced in next subsection.



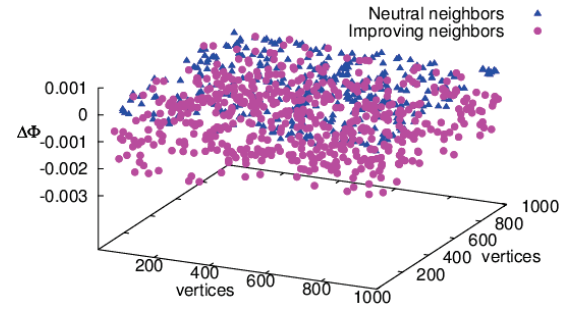
a) SD-LAalgorithm



b) SD-Φ algorithm



c) SD-LAalgorithm



d) SD-Φ algorithm

Figure 3 Comparison between the neutral and improving neighbors produced by SD-LA and SD-Φ over the *random*A1 instance, a) and b) after 1500 iterations, c) and d) at the iteration where SD-LA stops.

3.5.1 Neighborhood relations

The second neighborhood relation analyzed in this experiment permits to exchange the label of a vertex u with those of its adjacent vertices. It is formally defined in Eq. (17).

$$\mathcal{N}_2(\varphi)=\{\varphi' \in \mathcal{L}: swap(\varphi,u,v)=\varphi', u,v \in V, v \in A(u)\}\ , \quad (17)$$

where $A(u)$ is the set of adjacent vertices of u.

In order to bring forward the third studied neighborhood function some preliminary concepts should be presented. Given a vertex u with d adjacent vertices whose labels are $l_1,\ldots,l_d$, they can be sorted so that $s_1 < s_2 <\ldots< s_d$, where s_i is called the *i-th* order statistic [26]. Then the *statistical median* of the labels currently assigned to the vertices in $A(u)$ is given by Eq. (18):

$$median(u) = \begin{cases} s_{((d+1)/2)} & \text{If } d \text{ is odd} \\ \frac{1}{2}\left(s_{(d/2)} + s_{(1+d/2)}\right) & \text{If } d \text{ is even} \end{cases} \quad (18)$$

The neighborhood $\mathcal{N}_3(\varphi)$ of a labeling φ, presented in Eq. (19), employs this concept to find the most suitable choice for labeling a vertex u with respect to its adjacent vertices.

$$\mathcal{N}_3(\varphi) = \{\varphi' \in \mathcal{L}: swap(\varphi,u,v) = \varphi', u, v \in V, v \in M(u)\} \quad (19)$$

where $M(u)$ is a set containing those vertices whose current labels are close to the value $median(u)$ and that is formally defined as follows:

$$M(u) = \{v: median(u) - 2 \leq \varphi(u) \leq median(u) + 2\} \quad (20)$$

The fourth neighborhood relation $\mathcal{N}_4(\varphi)$ analyzed in this experiment is defined in Eq. (21):

$$\mathcal{N}_4(\varphi) = \{\varphi' \in \mathcal{L}: rotate(\varphi,i,j) = \varphi', i, j \in L, i < j\} \quad (21)$$

where $rotate(\varphi,i,j)$ is the product of $(j-i)swap$ operations (see Eq. (22)), $L = \{1,2,\ldots,n\}$ the set of labels of a graph of order n, and φ^{-1} a function associating to each label number k the vertex of the graph which contains it.

$$\begin{aligned} &rotate(\varphi,i,j) = \\ &swap\left(\varphi,\varphi^{-1}(i),\varphi^{-1}(j)\right) * \\ &swap\left(\varphi,\varphi^{-1}(i),\varphi^{-1}(j-1)\right) * \\ &swap\left(\varphi,\varphi^{-1}(i),\varphi^{-1}(j-2)\right) * \ldots * \\ &swap\left(\varphi,\varphi^{-1}(i),\varphi^{-1}(i+1)\right) \end{aligned} \quad (22)$$

Finally, the neighborhood function $\mathcal{N}_5(\varphi)$ is defined according to the following expression:

$$\mathcal{N}_5(\varphi) = \{\varphi' \in \mathcal{L}: inversion(\varphi,u,v) = \varphi', u, v \in V, u < v\} \quad (23)$$

where $inversion(\varphi,u,v)$ is the product of $\left\lfloor\frac{v-u}{2}\right\rfloor$ $swap$ operations as denoted in Eq. (24).

$$\begin{aligned} &inversion(\varphi,u,v) = \\ &swap(\varphi,u,v) * swap(\varphi,u+1,v-1) * \\ &swap(\varphi,u+2,v-2) * \ldots * \\ &swap\left(\varphi,u+\left\lfloor\frac{v-u}{2}\right\rfloor, v-\left\lfloor\frac{v-u}{2}\right\rfloor\right). \end{aligned} \quad (24)$$

3.5.2 Computational experiments

For these experiments we have used a slightly modified version of the SD algorithm presented in Section 3.1, which implements the best admissible move strategy with a restricted number of 2500 neighboring solutions in order to reduce the total expended computational time.

The 21 benchmark instances described in Section 3.2 were once again used. Similar results were obtained with all of them. However, with an aim of synthesizing the information, we present them by using only some representative graphs. All the results presented in this section are based on average data obtained in 100 independent runs with each of the ten combinations between the two evaluation functions and the five neighborhood relations.

The graph in Fig. 4a shows the convergence process, in terms of average solution quality achieved by the SD algorithm, when each of the ten studied combinations is used to solve the *random* A1 instance. In Figs. 4b to 4f each of the five neighborhood relations is analyzed individually. The following observations were made from these graphs.

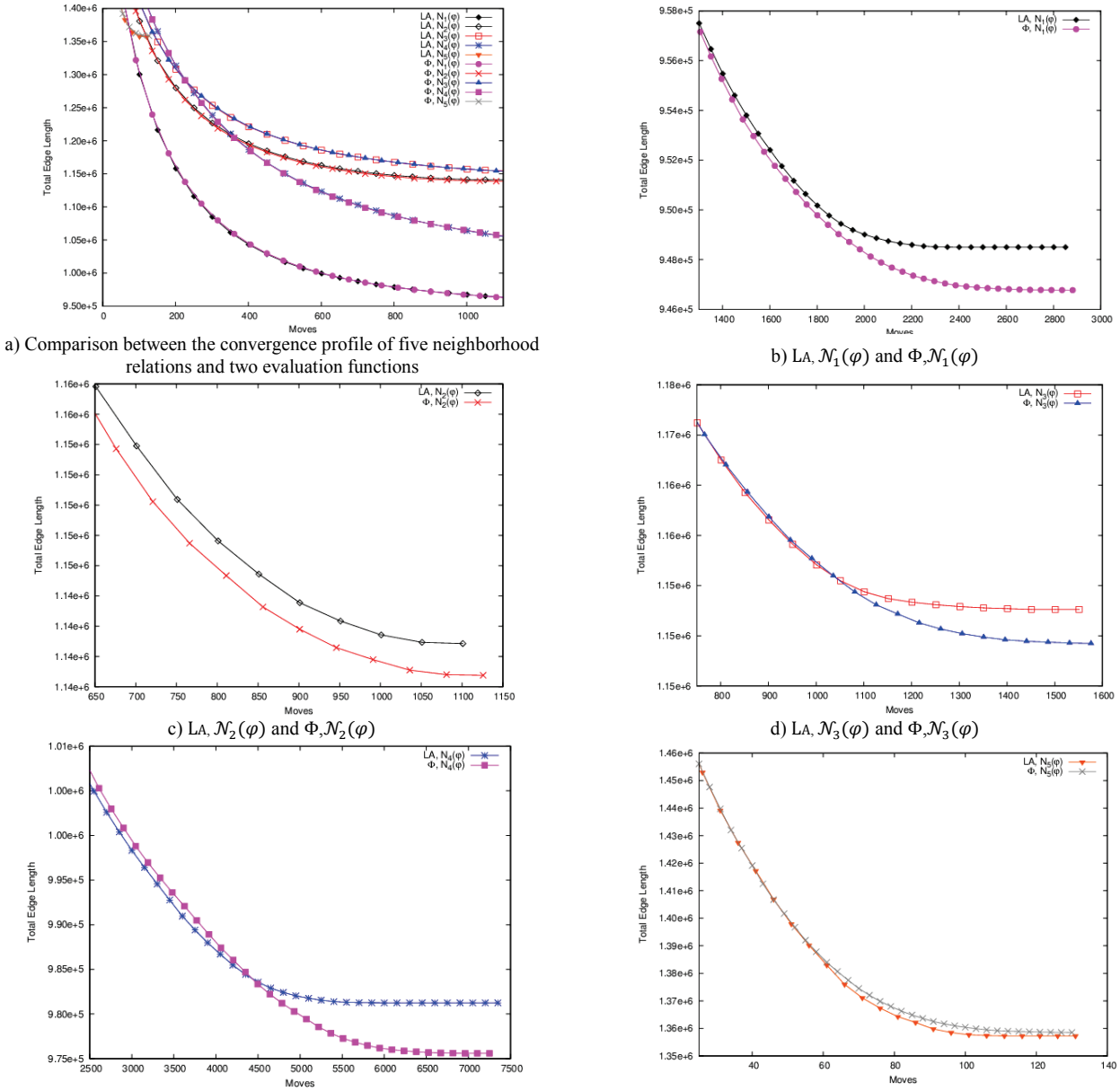
The Φ evaluation function allows, in four out of five cases, to improve the results provided by the SD algorithm, which highlights that Φ is more effective than LA. Indeed, Fig. 4f gives a clear example where a wrong choice of the neighborhood relation results into an algorithm with a poor performance, even if Φ is a better evaluation function than LA. We think that this behavior is due to the highly disruptive nature of $\mathcal{N}_5(\varphi)$.

With regard to the studied neighborhood relations, we have noticed from Figs. 4b to 4f that the best performance is reached by SD when the $\mathcal{N}_1(\varphi)$ neighborhood is used. The neighborhoods $\mathcal{N}_2(\varphi)$, $\mathcal{N}_3(\varphi)$ and $\mathcal{N}_5(\varphi)$ produce worse solutions than $\mathcal{N}_1(\varphi)$ and $\mathcal{N}_4(\varphi)$.

Finally, if we consider the ten analyzed combinations between the two evaluation functions and the five neighborhood relations, the best among them is when φ and $\mathcal{N}_1(\varphi)$ are used simultaneously (reaching an average solution quality $LA = 946\,774{,}368$ (see Tab. 2). The nine other combinations provide lower quality solutions. Based on the conclusions obtained from the current section, below we will present more computational results with two more advanced metaheuristic algorithms using as neighboring relation $\mathcal{N}_1(\varphi)$.

4 Comparing LA and Φ within other metaheuristics

After having studied the characteristics of Φ by using a simple Steepest Descent algorithm, we decided to evaluate the practical usefulness of Φ within two well-known and more elaborated metaheuristics. The results obtained from these experimental comparisons are provided in the following subsections.



a) Comparison between the convergence profile of five neighborhood relations and two evaluation functions

b) LA, $\mathcal{N}_1(\varphi)$ and Φ,$\mathcal{N}_1(\varphi)$

c) LA, $\mathcal{N}_2(\varphi)$ and Φ,$\mathcal{N}_2(\varphi)$

d) LA, $\mathcal{N}_3(\varphi)$ and Φ,$\mathcal{N}_3(\varphi)$

e) LA, $\mathcal{N}_4(\varphi)$ and Φ,$\mathcal{N}_4(\varphi)$

b) LA, $\mathcal{N}_5(\varphi)$ and Φ,$\mathcal{N}_5(\varphi)$

Figure 4 Analysis of the interaction between the neighborhood and evaluation functions and its influence on the performance of a SD algorithm over the *random* A1 instance.

Table 2 Performance comparison between ten combinations of neighborhood relations and evaluation functions over the *random* A1 instance

Evaluation Function	Neighborhood Relation				
	$\mathcal{N}_1(\varphi)$	$\mathcal{N}_2(\varphi)$	$\mathcal{N}_3(\varphi)$	$\mathcal{N}_4(\varphi)$	$\mathcal{N}_5(\varphi)$
LA	948 504,5	1 140 849,95	1 152 625,260	981 252,890	1 357 232,760
Φ	94 677,368	1 138 760,122	1 149 234,913	975 584,416	1 358 518,565

4.1 Iterated Local Search algorithm

For the purpose of the first experimental comparison of this section we have implemented a basic Iterated Local Search (ILS) algorithm [27, 28]. It was kept as simple as possible to obtain a clear idea of the evaluation function's influence over the performance of this metaheuristic algorithm. Our ILS implementation shares the following components with the SD algorithm presented in Section 3.1: permutation-based representation, evaluation functions LA and Φ, swap neighborhood relation $\mathcal{N}_1(\varphi)$ and random initial solution. It applies a local search method (embedded heuristic) to an initial solution φ_0 until it finds a local optimum, then this solution is perturbed and used as a starting point of another round of local search. After each local search the new local optimum solution found φ'' is accepted as the new incumbent solution φ^* if and only if its total edge length (cost) is better than that of φ^*. This iterative procedure repeats until a given stop condition is met. In the case of our ILS algorithm, the search process stops when the maximum number of non-improving neighboring solutions $maxNI$ allowed is reached (see Alg. 1).

A modified version of the SD algorithm presented in Section 3.1 was used as embedded heuristic; it implements the best admissible move strategy with a number of 2500 visited neighboring solutions. The

perturbation operator employed by our ILS consists in applying once the left rotation neighborhood relation $\mathcal{N}_4(\varphi)$ analyzed in Section 3.5.

```
Input: Neighborhood function 𝒩, evaluation function f,
       maximum non − improving neighboring solutions maxNI
φ0 ← GenerateInitialSolution()
φ* ← Local Search(φ0)
NI ← 0
while NI < maxNI do
   φ´ ← Perturbation(φ*)
   φ´´ ← LocalSearch(φ´)
   if f(φ´´) < f(φ*) then
      φ* ← φ´´
      NI ← 0
   else
      NI ← NI + 1
   end if
end while
return φ*
```

Algorithm 1 Iterated Local Search (ILS) algorithm

4.1.1 Computational experiments

For this experiment the ILS algorithm presented above was implemented. Let us call it ILS-LA or ILS-Φ according to whether evaluation function LA or Φ is used. In all the experiments carried out with this algorithm, the same test-suite and computational platform described in Section 3.2 were used. In order to guarantee as much as possible a fair comparison between both ILS algorithms, the same parameters (determined experimentally) were employed to compare them: a) initial solution randomly generated, b) swap neighborhood relation $\mathcal{N}_1(\varphi)$ for the embedded heuristic, c) left rotation neighborhood relation $\mathcal{N}_4(\varphi)$ for the perturbation operator and d) maximum non-improving neighboring solutions $maxNI = 10$. Due to the random nature of ILS, 100 independent runs were executed for each of the selected benchmark instances. When averaged results are reported, they are based on these 100 executions.

The results of this experiment are presented in Tab.3. Column one depicts the name of the graph. The next six columns display the best cost in terms of total edge length *C*, the average cost *Avg.* and the average CPU time *T* in seconds for both ILS-LA and ILS-Φ respectively. Column eight, Δ_C, presents the percentage gain in terms of the average costs reached by ILS-Φ compared to that produced by ILS-LA, i.e. $100 * (1$ - *average cost of* SD-Φ / *average cost of* SD-LA). Finally, Column nine lists the *p-value* obtained from a statistical significance analysis performed for the results of this experiment, according to the methodology described in Section 3.3.

It can be observed from Tab. 3 that ILS-Φ expends a slightly higher CPU time than ILS-LA, since it uses an average of 16,19 seconds for solving these 21 benchmark instances. On the contrary, ILS-LA employs only 12,19 seconds for this task. However, we can also remark that ILS-Φ can take advantage of its longer executions. Indeed, it is able to consistently improve the best results found by ILS-LA, obtaining in certain instances, like *bintree10*, an important decrease in total edge length (Δ_C up to 14,99%). Furthermore, the statistical analysis presented in the last column of Tab. 3 confirms that this increase in performance achieved by ILS-Φ with regard to ILS-LA is statistically significant on 80,95% of the studied instances (17 out of 21). It allows us to conclude that the Φ evaluation function leads to a more effective exploration of the search space than LA, since it does boost the performance of the proposed ILS algorithm.

Table 3 Performance comparison between ILS-LA and ILS-Φ

	ILS-LA			ILS-Φ			%	
Graph	*C*	*Avg.*	*T*	*C*	*Avg.*	*T*	Δ_C	*p-value*
randomA1	905 612	925 913,8	5,94	903 334	922 395,6	9,51	0,38	**2,70E-02**
randomA2	6 608 842	6 645 370,2	20,92	6 591 274	6 638 897,1	27,40	0,10	**1,70E-02**
randomA3	14 294 511	14 350 864,5	40,56	14 282 423	14 345 425,7	47,74	0,04	1,50E-01
randomA4	1 767 498	1 788 134,5	7,84	1 756 000	1 781 871,0	11,55	0,35	**3,30E-05**
randomG4	259 598	356 766,6	4,37	244 432	343 316,6	4,54	3,77	6,20E-02
bintree10	28 642	35 870,7	4,64	24 412	30 493,1	11,98	14,99	**5,70E-23**
hc10	533 912	594 226,7	6,56	525 184	585 474,4	4,73	1,47	9,90E-02
mesh33x33	59 035	90 808,0	7,59	45 110	79 239,9	14,85	12,74	**2,40E-08**
3elt	1 740 660	2 245 592,6	33,81	1 540 158	2 165 033,8	41,28	3,59	**1,30E-02**
airfoil1	1 402 825	1 821 485,3	30,3	1 132 536	1 642 201,2	39,43	9,84	**1,10E-08**
whitaker3	9 231 695	11 010 927,5	59,36	9 161 792	10 750 209,2	66,66	2,37	**3,20E-02**
c1y	87 559	100 558,0	4,73	80 898	96 113,8	8,46	4,42	**1,40E-04**
c2y	112 621	135 336,1	5,32	109 764	126 667,0	9,65	6,41	**2,30E-09**
c3y	192 372	234 044,5	6,94	189 398	225 294,8	10,32	3,74	**2,40E-03**
c4y	175 723	236 348,9	6,98	169 899	211 838,5	12,39	10,37	**3,70E-09**
c5y	157 201	187 924,7	6,23	145 286	175 624,7	11,71	6,55	**1,80E-07**
gd95c	520	657,4	0,14	516	649,3	0,16	1,22	3,50E-01
gd96a	127 957	138 367,2	3,12	121 476	132 382,0	6,19	4,33	**2,80E-20**
gd96b	1486	1742,3	0,12	1475	1631	0,17	6,39	**6,60E-05**
gd96c	524	616,9	0,11	520	576,5	0,27	6,55	**8,00E-04**
gd96d	2776	3425,7	0,38	2528	3070,4	1,11	10,37	**1,30E-07**
Average	1 794 836,6	1 947 856,3	12,19	1 763 257,9	1 917 066,9	16,19	5,24	17+

4.2 Tabu Search algorithm

For the second experimental comparison of the studied evaluation functions we have selected the Tabu Search (TS) algorithm [29], since it is among the most cited and used metaheuristics for solving combinatorial optimization problems [30].

The pseudo-code of our TS implementation is presented in Alg. 2. It starts with a randomly generated solution, then it proceeds iteratively to visit a series of locally best configurations following the neighborhood

function $\mathcal{N}_1(\varphi)$. At each iteration, a best neighbor φ' is chosen to replace the current configuration φ, even if the former does not improve the current one. In order to explore consecutive local optimal solutions and to avoid the occurrence of cycles, TS introduces the notion of tabu list. The basic idea is to record the attributes of each visited solution and to forbid the algorithm to visit again this configuration during the next *Tt* iterations (*Tt* is called the tabu tenure).

```
Input: Neighborhood function N, evaluation function f,
       maximum non − improving neighboring solutions maxNI
φ0 ← GenerateInitialSolution()
φ* ← φ
InitializeTabuList()
NI ← 0
while stop condition not met do
   φ' ← ChooseBestAdmissible(φ)
        // {φ' ∈ N(φ)|φ' non − tabu or aspiration condition holds}
   UpdateTabuListAndAspirantionCondition()
   φ ← φ'
   if f(φ) < f(φ*) then
      φ* ← φ
      NI ← 0
   else
      NI ← NI + 1
   end if
   if NI > maxNI then Diversification(φ)
end while
return φ*
```

Algorithm 2 Tabu Search (TS) algorithm

In our TS algorithm the neighbor of a given solution φ is obtained by swapping the labels of any pair (i, j) of different vertices. When such a move is performed the couple of vertices (i, j) is classified tabu for the next *Tt* iterations. Therefore, the vertices i and j cannot be exchanged during this period. Nevertheless, a tabu move leading to a configuration better than the best configuration found so far φ^* is always accepted (aspiration criterion).

The tabu tenure *Tt* for a move, in our TS algorithm, is dynamically calculated during the search using the approach introduced by Galinier [31] and used later in [32]. It is based on the use of a periodic step function *PS* which takes as argument the number of iterations *iter*. Each period of this function is composed of 1500 iterations divided into 15 intervals $[x_i, x_{i+1} - 1]_{i=1,2,\ldots,15}$ with $x_1 = 1$ and $x_{i+1} = x_i + 100$. The value returned by *PS* for a particular iteration *iter*$\in [x_i, x_{i+1} - 1]$ is given by $(a_i)_{i=1,2,\ldots,15} = (1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1) \times \alpha$, where α is a parameter that fixes the minimum tenure value. Therefore, the tabu tenure equals α between iterations 1 and 99, $2 \times \alpha$ between iterations 100 and 199, followed by α again for iterations [200,299] and $4 \times \alpha$ for iterations [300,399], etc. This variation scheme is periodically repeated by this function after every 1500 iterations.

A diversification mechanism was also implemented since the above basic TS algorithm could be trapped in deep local optima. In our case, the search is judged stagnating each time the best solution found so far φ^* is not further improved after a number *NI* of consecutive iterations. To help the search to escape from such deep local optima, we apply a simple perturbation mechanism to the current solution to bring diversification into the search. The perturbation consists in applying λ consecutive times the left rotation neighborhood relation $\mathcal{N}_4(\varphi)$ analyzed in Section 3.5, where λ is a parameter that fixes the strength of the perturbation. Finally, our TS algorithm stops when it ceases to make progress, i.e., when *MD* successive diversification iterations do not produce a better solution.

4.2.1 Computational experiments

Two versions of the TS algorithm previously presented were implemented. Let us call them TS-LA or TS-Φ depending on which evaluation function is employed. The following parameters were determined experimentally for both TS algorithms, and used consistently for this experiment: a) initial solution randomly generated, b) swap neighborhood relation $\mathcal{N}_1(\varphi)$ with a maximum number of 2500 neighboring solutions, c) maximum non-improving neighboring solutions *maxNI*=100, d) minimum tenure value $\alpha = 15$, e) the strength of the diversification operator is $\lambda = 2$, f) maximum non-improving successive diversification iterations *MD*=20. Due to the random nature of the TS algorithm, 100 independent runs were executed for each of the selected benchmark instances.

Tab. 4 summarizes the results obtained from this experiment. Columns two to seven display the best cost in terms of total edge length *C*, the average cost *Avg.* and the average CPU time *T* in seconds achieved by TS-LA and TS-Φ respectively. Column eight, Δ_C, depicts the percentage gain in terms of the average costs reached by TS-Φ compared to that produced by TS-LA, while column nine presents the *p-value* obtained from a statistical significance analysis performed for the results of this experiment using the same methodology described in Section 3.3.

The analysis of the data presented in Tab. 4 leads us to the following observations. First, we can notice that TS-Φ consumes slightly more computing time than TS-LA (in average 199,78 vs. 194,85 seconds). Second, we clearly remark that TS-Φ consistently returns solutions of better quality than TS-LA, since in average TS-Φ provides solutions whose total edge lengths are $\Delta_C = 6{,}15\%$ smaller than those produced by TS- LA. For certain instances, like *whitaker3*, the decrease in total edge length could be up to $\Delta_C = 31{,}71$. Third, the statistical analysis whose results are presented in the last column of Tab. 4 (*p-value*) confirms that the increase in performance achieved by TS-Φ with respect to TS-LA is statistically significant on 80,95% of the studied instances (17 out of 21).

Thus, this second experiment confirms that Φ does help the TS algorithm to make a more effective search than LA.

It should be clear that given the algorithm-independent nature of the Φ evaluation function, it can be used by other advanced metaheuristics for the MinLA problem to boost the search performance. Finally, let us mention that Φ is one of the key components that contributes to the performance of a highly successful two-stage SA algorithm presented in [12].

Table 4 Performance comparison between TS-LA and TS-Φ.

	TS-LA			TS-Φ			%	
Graph	*C*	*Avg.*	*T*	*C*	*Avg.*	*T*	Δ_C	*p-value*
randomA1	894048	911156,8	61,53	890615	907896,5	61,21	0,36	**4,90E-03**
randomA2	6587805	6629381,6	136,75	6576500	6618055,2	203,02	0,17	**1,10E-05**
randomA3	14289228	14331958,2	248,05	14273622	14327253,5	276,79	0,03	1,50E-01
randomA4	1747516	1774152,9	66,75	1745643	1768267,3	98,34	0,33	**1,50E-05**
randomG4	223838	332724,2	26,43	202757	317200,9	15,32	4,67	**7,30E-03**
bintree10	18055	24683	73,95	16352	23969,8	106,82	2,89	1,90E-01
hc10	526468	582513,5	27,53	524320	571831,9	17,19	1,83	**1,80E-02**
mesh33x33	33805	55991,9	82,66	32331	46434,3	89,16	17,07	**3,20E-07**
3elt	635048	1075592,1	525,06	523352	900929,3	537,95	16,24	**1,60E-09**
airfoil1	436307	794070	473,03	433859	682673,2	420,61	14,03	**1,50E-07**
whitaker3	2079351	4190127,1	1887,45	1239421	2861347,2	1794,89	31,71	**9,80E-25**
c1y	66626	86193,9	59,27	65816	82268,6	86,37	4,55	**3,30E-04**
c2y	87939	111234,7	64,31	83883	105144,7	72,02	5,47	**1,80E-05**
c3y	146165	184244,1	99,69	140866	173305,5	129,74	5,94	**1,40E-06**
c4y	136706	175863,4	112,83	132090	162776,4	62,28	7,44	**2,10E-05**
c5y	120029	146903,5	89,96	113807	140538,9	153,65	4,33	**1,10E-03**
gd95c	518	624,6	0,49	509	615	0,37	1,53	4,20E-01
gd96a	113045	122945,1	50,62	111671	121649,6	63	1,05	**1,20E-02**
gd96b	1463	1705,6	0,68	1461	1647,4	1,13	3,41	2,20E-01
gd96c	520	554,8	0,86	519	547,5	0,28	1,33	**4,00E-03**
gd96d	2634	3087,7	4,03	2520	2940,5	5,3	4,77	**1,20E-04**
Average	1340338,8	1501700,4	194,85	1291043,5	1419871,1	199,78	6,15	17+

5 Some related works

There are relatively few studies on more discriminating evaluation functions for metaheuristics. Two notable examples are reported for graph coloring [7] and for bin packing [33].

The *Graph Coloring Problem* (GCP) consists in coloring the vertices of a given graph with a minimal number of colors (chromatic number) with the constraint that two adjacent vertices receive different colors. For this problem, Johnson *et al*. proposed the evaluation function g depicted in Eq. (25). In this equation the number of colors k is considered variable, V_i represents the set of vertices colored with i and E_i $(1 \leq i \leq k)$ is the set of edges both of whose endpoints (vertices) are in V_i.

$$g(x) = -\sum_{i=1}^{k} |V_i|^2 + \sum_{i=1}^{k} 2|V_i||E_i| \tag{25}$$

Observe that by the first term of Eq. (25), a large color set V_i tends to get more vertices than smaller ones, and as a side effect of this, the number of colors k is minimized. The second term is used to penalize those edges having the same colors on the end vertices. Another important characteristic is that all the local optima under g correspond to legal colorings (i.e., $E_i = \emptyset \; \forall \; i$). But the main advantage of the g evaluation function, compared with the objective function f (i.e., the weighted sum of k and $\sum_{i=1}^{k} |E_i|$), is that it can evaluate the gain of a move in which the number of colors k is not changed but the size of a small V_i is decreased. As it was demonstrated by Johnson *et al.*[7], the use of the g evaluation function within a SA algorithm leads to good results, because g is more informative than f and helps to better guide the search process.

The *Bin Packing Problem* consists in packing, within a minimum number of bins of a given capacity Q, a set of n items $A = \{a_1, a_2, \ldots, a_n\}$, each having a size $s(a_i) > 0$. In other words, the goal is to pack the items into as few bins as possible, i.e., partition them into a minimum number m of subsets $B_1, B_2, \ldots, B_m$ such that for each $B_j, \sum_{a_i \in B_j} s(a_i) \leq Q$. The first evaluation function which comes to mind is the number of bins used to pack all the items, but it is unusable in practice because it is not informative to guide an algorithm in the search. Falkenauer [33] proposed instead the following evaluation function (to maximize):

$$g(x) = \frac{\sum_{i=1}^{m} (F_i/Q)^k}{m}, \tag{26}$$

where m is the number of bins used in the solution x, F_i the sum of sizes of the items packed in the bin i (the fill of the bin) and k a constant $(k > 1)$, which expresses the concentration on the most filled bins in comparison to the less filled ones. The larger k is, the more well-filled bins will be produced.

The proposed evaluation function g maximizes the average bin efficiency (over all bins) measuring the exploitation of a bin's capacity. Thus, it encourages the bin efficiency, rather than the overall performance of all the bins together. Additionally, g assigns similar (but not equal) values to similar solutions, while having the same optima as the objective function f, producing as consequence smoother search landscapes.

Other recent examples of more discriminating evaluation functions can be found in [34, 35].

6 Conclusions and discussions

Evaluation function is one fundamental element of a metaheuristic algorithm. By extending a preliminary work reported in [13], this paper described an in-depth investigation of the notion of evaluation function using a well-known graph labeling problem (i.e., the Minimum Linear Arrangement problem, MinLA) as a representative case of study.

Some important mathematical properties of the conventional MinLA evaluation function LA (i.e., total edge length cost) were throughly analyzed. The results from this study highlighted the potential drawbacks of

using this function to guide the search and enabled us to devise a more discriminating evaluation function, namely Φ. The basic idea behind Φ is to integrate in the evaluation function not only the total edge length of an arrangement (L_A), but also other semantic information induced by the absolute differences between the labels assigned to adjacent nodes of the graph.

The practical usefulness of the evaluation function Φ was first assessed with a parameter-free Steepest Descent (SD) algorithm using a full test-suite composed of the 21 well-known benchmarks of the literature [8, 9, 10, 11, 12]. The results produced by this experiment showed that the SD algorithm using Φ as evaluation function (SD-Φ) returns for 19 out of 21 instances better results than those produced by SD-L_A requiring in average a comparable CPU time per iteration. The statistical analysis of these results confirmed that the increase in performance achieved by SD-Φ with regard to SD-L_A is statistically significant on 66,66% of the studied instances (14 out of 21).

In order to get a better understanding of these results an experimental analysis of the search landscapes and the distributions of improving neighbors induced by these two evaluation functions was carried out. It provided experimental evidences showing that the superiority of Φ over L_A, for guiding the search, is due to its ability to identify in average a greater number of improving neighbors than L_A.

To gain more insights into the real working of the studied evaluation functions a second experimental comparison between them was carried out employing two more elaborated metaheuristics: Iterated Local Search (ILS) and Tabu Search (TS). The analysis of the data produced by these comparisons showed that the performance of the ILS and TS metaheuristics can be boosted by using more discriminating evaluation functions like Φ. Indeed, ILS-Φ and TS-Φ were able to consistently improve the best results produced by ILS-L_A and TS-L_A, respectively. Furthermore, the statistical analysis carried out over these results demonstrated that the increase in performance achieved by the metaheuristics using Φ as evaluation function with respect to those employing L_A is statistically significant on 80,95% of the studied instances (17 out of 21). These results allow us to conclude that the Φ evaluation function permits the search algorithm using it to make a more effective exploration of the search space than L_A.

Even if the current stage of our knowledge does not allow us to identify general rules for designing more informative evaluation functions, we hope the work reported in this paper sheds useful light on the way that may be followed. We also expect the results shown in this work incite more research on more discriminating evaluation functions as an effective mean of boosting the performance of metaheuristic algorithms.

Acknowledgements

This research was partially supported by projects No. 105060 and 99276 from The National Council of Science and Technology of Mexico (CONACyT). The authors would like to thank the anonymous reviewers for their valuable feedback that greatly contributed to improving this paper.

7 References

[1] Gendreau, M.; Potvin, J. Y. (Editors). Handbook of Metaheuristics (International Series in Operations Research & Management Science). Springer, 2010.

[2] Hoos, H. H.; Stützle, T. Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, 2004.

[3] Mladenović, N.; Hansen, P. Variable Neighborhood Search. // Computers & Operations Research. 24, 11(1997), pp. 1097-1100.

[4] Gaspero, L. D; Schaerf, A. Neighborhood Portfolio Approach for Local Search Applied to Timetabling problems. // Journal of Mathematical Modeling and Algorithms. 5, 1(2006), pp. 65-89.

[5] Goëffon, A.; Richer, J. M.; Hao, J. K. Progressive Tree Neighborhood Applied to the Maximum Parsimony Problem. // IEEE/ACM Transactions on Computational Biology and Bioinformatics. 5, 1(2008), pp. 136-145.

[6] Stadler, P. F. Correlation in Landscapes of Combinatorial Optimization Problems. // Europhysics Letters. 20, (1992), pp. 479-482.

[7] Johnson, D. S.; Aragon, C. R.; McGeoch, L. A.; Schevon, C. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. // Operations Research. 39, 3(1991), pp. 378-406.

[8] Petit, J. Experiments on the Minimum Linear Arrangement Problem. // The ACM Journal of Experimental Algorithmics, 8, (2003), DOI: http://dx.doi.org/10.1145/996546.996554.

[9] Bar-Yehuda, R.; Even, G.; Feldman, J.; Naor, J. Computing an Optimal Orientation of a Balanced Decomposition Tree for Linear Arrangement Problems. // Journal of Graph Algorithms and Applications. 5, 4(2001), pp. 1-27.

[10] Koren, Y.; Harel, D. A Multi-Scale Algorithm for the Linear Arrangement Problem // Lecture Notes in Computer Science. 2573, (2002), pp. 293-306.

[11] Rodriguez-Tello, E.; Hao, J. K.; Torres-Jimenez, J. Memetic Algorithms for the MinLA Problem. // Lecture Notes in Computer Science. 3871, (2006), pp. 73-84.

[12] Rodriguez-Tello, E.; Hao, J. K.; Torres-Jimenez, J. An Effective Two-Stage Simulated Annealing Algorithm for the Minimum Linear Arrangement Problem. // Computers & Operations Research. 35, 10(2008), pp. 3331-3346.

[13] Rodriguez-Tello, E.; Hao, J. K.; Torres-Jimenez, J. A Refined Evaluation Function for the MinLA Problem. // Lecture Notes in Computer Science. 4293, (2006), pp. 392-403.

[14] Harper, L. H. Optimal Assignment of Numbers to Vertices. // SIAM Journal on Applied Mathematics. 12, 1(1964), pp. 131-135.

[15] Adolphson, D. L.; Hu, T. C. Optimal Linear Ordering. // SIAM Journal on Applied Mathematics. 25, 3(1973), pp. 403-423.

[16] Mitchison, G.; Durbin, R. Optimal Numbering of an N×N Array. // SIAM Journal on Matrix Analysis and Applications. 7, 4(1986), pp. 571-582.

[17] Karp, R. M. Mapping the Genome: Some Combinatorial Problems Arising in Molecular Biology. // ACM Press, Proceedings of the 25th annual ACM symposium on Theory of computing / San Diego, CA, USA, 1993, pp. 278-285.

[18] Ravi, R.; Agrawal, A.; Klein, P. N. Ordering Problems Approximated: Single-Processor Scheduling and Interval Graph Completion. // Lecture Notes in Computer Science. 510, (1991), pp. 751-762.

[19] Shahrokhi, F.; Sykora, O.; Szekely, L. A.; Vrto, I. On Bipartite Drawings and the Linear Arrangement Problem. // SIAM Journal on Computing. 30, 6(2001), pp. 1773-1789.
[20] Lai, Y. L.; Williams, K. A Survey of Solved Problems and Applications on Bandwidth, Edgesum, and Profile of Graphs. // Graph Theory. 31, (1999), pp.75-94.
[21] Lin, S.; Kernighan, B. W. An Effective Heuristic Algorithm for the Traveling Salesman Problem. // Operations Research. 21, 2(1973), pp. 498-516.
[22] Garey, M. R.; Johnson, D. S. Computers and Intractability: A guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York, 1979.
[23] Safro, I.; Ron, D.; Brandt, A. Graph Minimum Linear Arrangement by Multilevel Weighted Edge Contractions. // Journal of Algorithms. 60, 1(2006), pp. 24-41.
[24] Rotman, J. J. Advanced Modern Algebra. Prentice Hall, 2003.
[25] Lü, Z.; Hao, J. K.; Glover, F. Neighborhood Analysis: A CaseStudy on Curriculum-based Course. // Journal of Heuristics. 17, 2(2011), pp. 97-118.
[26] Larsen, R. J.; Morris, L. M. An Introduction to Mathematical Statistics and Its Applications. 4th edition Upper Saddle River, Prentice Hall , NJ, USA, 2005.
[27] Martin, O.; Otto, S. W.; Felten, E. W. Large-step Markov Chains for the Traveling Salesman Problem. // Complex Systems. 5, 3(1991), pp. 299-326.
[28] Lourenco, H. R.; Martin, O.; Stützle, T. Iterated Local Search. // Handbook of Metaheuristics, International Series in Operations Research & Management Science, Kluwer Academic Publishers, 2002, pp. 321-353.
[29] Glover, F.; Laguna, M. Tabu Search. Kluwer AcademicPublishers, 1997.
[30] Blum, C.; Roli, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. // ACM Computing Surveys. 35, 3(2003), pp. 268-308.
[31] Galinier, P.; Boujbel, Z.; Coutinho-Fernandes, M. An Efficient Memetic Algorithm for the Graph Partitioning Problem. // Annals of Operations Research. 191, 1(2011), pp. 1-22.
[32] Wu, Q.; Hao, J. K. Memetic Search for the Max-bisection Problem. // Computers & Operations Research. 40, 1(2013), pp. 166-179.
[33] Falkenauer, E. A Hybrid Grouping Genetic Algorithm for Bin Packing. // Journal of Heuristics. 2, (1996), pp. 5-30.
[34] Rodriguez-Tello, E.; Hao, J. K.; Torres-Jimenez, J. An Improved Simulated Annealing Algorithm for Bandwidth Minimization. // European Journal of Operational Research. 185, 3(2008), pp. 1319-1335.
[35] Porumbel, D. C.; Hao, J. K.; Kuntz, P. A study of evaluation functions for the graph k-coloring problem. // Lecture Notes in Computer Science. 4926, (2007), pp. 124-135.

Authors' addresses

Eduardo Rodriguez-Tello, PhD.
CINVESTAV-Tamaulipas, Information Technology Laboratory,
Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria
Tamps., Mexico
E-mail: ertello@tamps.cinvestav.mx

Jin-Kao Hao, PhD.
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers
Cedex 01, France
E-mail: Jin-Kao.Hao@univ-angers.fr

Hillel Romero-Monsivais, MSc.
CINVESTAV-Tamaulipas, Information Technology Laboratory,
Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria
Tamps., Mexico
E-mail: hromero@tamps.cinvestav.mx