A Refined Evaluation Function for the MinLA Problem

Eduardo Rodriguez-Tello[1], Jin-Kao Hao[1], and Jose Torres-Jimenez[2]

[1] LERIA, Université d'Angers.
2 Boulevard Lavoisier, 49045 Angers, France
{ertello, hao}@info.univ-angers.fr
[2] Mathematics Department, University of Guerrero.
54 Carlos E. Adame, 39650 Acapulco Guerrero, Mexico
jose.torres.jimenez@acm.org

Abstract. This paper introduces a refined evaluation function, called Φ, for the Minimum Linear Arrangement problem (MinLA). Compared with the classical evaluation function (LA), Φ integrates additional information contained in an arrangement to distinguish arrangements with the same LA value. The main characteristics of Φ are analyzed and its practical usefulness is assessed within both a Steepest Descent (SD) algorithm and a Memetic Algorithm (MA). Experiments show that the use of Φ allows to boost the performance of SD and MA, leading to the improvement on some previous best known solutions.

Key words: Genetic Algorithms, Evaluation Function, Linear Arrangement, Heuristics.

1 Introduction

The evaluation function is one of the key elements for the success of evolutionary algorithms and more generally, heuristic search algorithms. It is the evaluation function that guides the search process toward good solutions in a combinatorial search space. The more discriminating 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 [3, 8]. It is also effective to dynamically change the evaluation function during the search, like in the noising method [1] and the search space smoothing method [6]. Another technique consists in developing new more informative evaluation functions which may not be directly related to the objective function such in [8, 9].

In this paper, we are interested in devising a refined evaluation function for the *Minimum Linear Arrangement* problem (MinLA). MinLA was first stated by Harper in [7]. His aim was to design error-correcting codes with minimal average absolute errors on certain classes of graphs. MinLA arises also in other research

fields like biological applications, graph drawing, VLSI layout and software diagram layout [2, 11].

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 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 arrangement φ is defined according to Equation 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.

MinLA is known to be NP-hard for general graphs [4], though there exist polynomial cases such as trees, rooted trees, hypercubes, meshes, outerplanar graphs, and others (see [2] for a detailed survey). To tackle the MinLA problem, a number of heuristic algorithms have been developed. Among these algorithms are a) heuristics especially developed for MinLA, such as the multi-scale algorithm [10] and the algebraic multi-grid scheme [14]; and b) metaheuristics such as Simulated Annealing [12] and Memetic Algorithms [13].

All these algorithms evaluate the quality of a solution (linear arrangement) as the change in the objective function $LA(G, \varphi)$. However, using LA as the evaluation function of a search algorithm represents a potential drawback. Indeed, different linear arrangements can have the same total edge length and can not be distinguished by LA, even though they do not have the same chances to be further improved.

In this paper, a more discriminating evaluation function (namely Φ) is proposed. The basic idea is to integrate in the evaluation function not only the total edge length of an arrangement (LA), but also other semantic information related to the arrangement.

The new evaluation function Φ is experimentally assessed regarding to the conventional LA evaluation function within both a Steepest Descent (SD) algorithm and a Memetic Algorithm (MA) by employing a set of benchmark instances taken from the literature. The computational results show that thanks to the use of Φ, the performance of the search algorithms is greatly improved, leading to the improvement on some previous best known solutions.

The reminder of this work is organized as follows. In Section 2, after analyzing the drawbacks of the classic evaluation function for MinLA, the refined evaluation function is formally described. Then, in Section 3, with the help of a parameter-free SD algorithm, the proposed evaluation function is assessed with respect to the conventional LA function. Additional analysis and comparisons are given in Section 4 within a Memetic Algorithm framework. Finally, Section 5 summarizes the contributions of this paper.

2 A Refined Evaluation Function for MinLA

The choice of the evaluation function (fitness function) is a very important aspect of any search procedure. Firstly, in order to efficiently test each potential solution, the evaluation function must be as simple as possible. Secondly, it must be sensitive enough to locate promising search regions on the space of solutions. Finally, the evaluation function must be consistent: a solution that has a higher probability for further improvement should get a better evaluation value.

The classical evaluation function for MinLA (LA) does not fulfill these requirements. In the next subsection LA's deficiencies are analyzed and a refined evaluation function is formally introduced.

2.1 The Φ Evaluation Function

A particular resulting value of the LA evaluation function can also be expressed by the Formula 2, where d_k refers to the appearing frequency of an absolute difference with value k between two adjacent vertices of the graph (see Table 1 for an example).

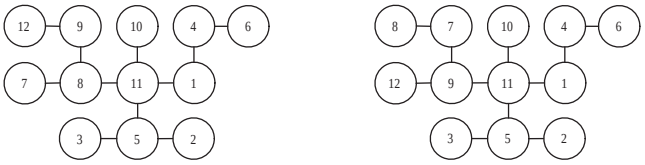
$$LA(G, \varphi) = \sum_{k=1}^{n-1} k d_k \tag{2}$$

This way of computing the solution quality is not sensitive enough to locate promising search regions on the space of solutions, because it does not make distinctions among the absolute differences (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 total edge length it is impossible to know which one has higher possibility for further improvement. For example, the two arrangements for the graph showed in Fig. 1 have the same cost ($LA = 35$), but one of them has in fact better chances to be improved in subsequent iterations of the search process. This point will be made clear below.

The Φ evaluation function that will be introduced below helps to overcome these disadvantages. This new 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 absolute differences of the graph. Furthermore, it maintains the fact that $\lfloor \Phi \rfloor$ results into the same integer value produced by Equations 1 and 2.

The main idea of Φ is to penalize the absolute differences having small values of k and to favor those with values of k near to the bandwidth β of the graph[1]. 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 frequency d_k should have a different contribution, which can be computed by

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



(a) (b)

Fig. 1. (a) Arrangement φ with $LA = 35$. (b) Arrangement φ' with $LA = 35$.

employing Equation 3.

$$k + \frac{1}{\prod_{j=1}^{k}(n+j)} = k + \frac{n!}{(n+k)!} \tag{3}$$

Then, the quality of an arrangement can be defined by the following expression:

$$\sum_{k=1}^{n-1} d_k \left(k + \frac{n!}{(n+k)!} \right) \tag{4}$$

By simplifying this formula we obtain the Equation 5, which represents the new Φ evaluation function. Observe that the first term in this formula is equal to Equation 2. The second term (a fractional value) is the discriminator for arrangements having the same LA value.

$$\Phi(G, \varphi) = \sum_{k=1}^{n-1} k d_k + \sum_{k=1}^{n-1} \frac{n! d_k}{(n+k)!} \tag{5}$$

In the following subsection the calculation of Φ is illustrated with an example.

2.2 A Calculation Example of the Φ Evaluation Function

Let us consider the two arrangements of the graph depicted in Fig. 1. In Table 1 the steps used in the computation of the second term in Equation 5, for each arrangement, are displayed. The rows corresponding to $d_k = 0$ were omitted because they do not contribute to the final result.

Then, by making the substitution of the resulting values in the Formula 5 we obtain: $\Phi(G, \varphi) = 35+2.43\text{E-}01 = 35.243$. In contrast if Φ is computed for φ' of Fig. 1(b), a different and smaller value is obtained: $\Phi(G, \varphi') = 35+1.77\text{E-}01 = 35.177$. It means that the arrangement φ' is potentially better than φ. Indeed it is better because it is easier to reduce the 4 absolute differences with value 2 ($d_2 = 4$) in φ' than the 3 absolute differences with value 1 ($d_1 = 3$) in φ.

In this sense Φ is more discriminating than LA and leads to smoother landscapes of the search process.

Table 1. Calculation of Φ for the two arrangements presented in Fig. 1.

	φ				φ'			
k	d_k	$n!d_k$	$(n+k)!$	$n!d_k/(n+k)!$	d_k	$n!d_k$	$(n+k)!$	$n!d_k/(n+k)!$
1	3	1.44E+09	6.23E+09	2.31E-01	2	9.58E+08	6.23E+09	1.54E-01
2	2	9.58E+08	8.72E+10	1.10E-02	4	1.92E+09	8.72E+10	2.20E-02
3	4	1.92E+09	1.31E+12	1.47E-03	3	1.44E+09	1.31E+12	1.10E-03
6	1	4.79E+08	6.40E+15	7.48E-08	1	4.79E+08	6.40E+15	7.48E-08
10	1	4.79E+08	1.12E+21	4.26E-13	1	4.79E+08	1.12E+21	4.26E-13
		Sum		2.43E-01				1.77E-01

2.3 Computational Considerations

In order to compute the quality of a linear arrangement φ by using the conventional LA evaluation function, we must calculate the sum $\sum_{(u,v)\in E} |\varphi(u)-\varphi(v)|$. Then it requires $O(|E|)$ instructions.

On the other hand, to efficiently compute the Φ evaluation function we could precalculate each term $k + (n!/(n+k)!)$ in the Equation 4 and store them in an array W. All this needs to execute $O(|V| + |V|)$ operations. Then each time that we need to calculate the value of Φ the sum $\sum_{(u,v)\in E} W[|\varphi(u) - \varphi(v)|]$ must be computed, which results into the same computational complexity as the one that is required to compute LA. Additionally, the Φ evaluation function allows an incremental cost evaluation of neighboring solutions (see Subsection 3.1). Suppose that the labels of two different vertices (u, v) are swapped, then we should only recompute the $|N(u)| + |N(v)|$ absolute differences that change, where $|N(u)|$ and $|N(v)|$ represent the number of adjacent vertices to u and v respectively. As it can be seen this is faster than $O(|E|)$.

In the next sections, we will carry out experimental studies in order to assess the effectiveness of the proposed evaluation function compared with the conventional LA function. This is realized first with a parameter free descent algorithm and then with a memetic algorithm.

3 Comparing the Evaluation Functions within a Steepest Descent Algorithm

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 allows a direct comparison of the two evaluation functions without bias. Next, the implementation details of this algorithm are presented.

Search Space, Representation and Fitness Function. For a graph G with n vertices, the search space $\mathcal{A}$ is composed of all $n!$ possible linear arrangements. In our SD algorithm a linear arrangement φ is represented 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 fitness of an arrangement φ is evaluated by using either the LA or Φ evaluation function (Equation 1 or 5 respectively).

Table 2. Performance comparison between SD-LA and SD-Φ.

		SD-LA			SD-Φ				
Graph	$\|V\|$	I	C	T	I	C	T	IN	Δ_C
randomA1	1000	2115.7	946033.1	0.0195	3134.2	941677.7	0.0182	741.5	-4355.4
randomA3	1000	2460.1	14397879.8	0.4839	2887.3	14396580.9	0.4681	625.1	-1298.8
bintree10	1023	1233.6	51716.8	0.0132	1406.9	51548.8	0.0132	229.4	-167.9
mesh33x33	1089	2994.4	130751.3	0.0153	8143.9	112171.7	0.0151	1347.6	-18579.6
3elt	4720	27118.4	2630144.6	0.2708	52061.6	2392981.3	0.2797	5616.7	-237163.3
airfoil1	4253	23566.4	2184693.3	0.2386	45909.1	1958983.4	0.2263	4575.9	-225709.9
c2y	980	2140.9	169434.7	0.0159	3078.9	167127.9	0.0171	578.3	-2306.8
c3y	1327	3334.1	282818.5	0.0559	5097.4	275529.3	0.0461	1008.8	-7289.2
gd95c	62	69.2	716.3	0.0002	81.3	697.4	0.0002	19.4	-18.9
gd96a	1096	2215.7	148158.3	0.0167	3283.5	144751.7	0.0177	804.3	-3406.6
Average									-50029.6

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

Neighborhood Function. The *neighborhood* $N(\varphi)$ of an arrangement φ is such that for each $\varphi \in \mathcal{A}$, $\varphi' \in N(\varphi)$ if and only if φ' can be obtained by swapping the labels of any pair of different vertices (u, v) from φ. This neighborhood is small and allows an incremental cost evaluation of neighboring solutions.

General Procedure. The SD algorithm starts from the initial solution $\varphi \in \mathcal{A}$ and repeats replacing φ with the best solution in its neighborhood $N(\varphi)$ until no better arrangement is found.

3.2 Computational Experiments

The purpose of this experiment is to study the characteristics of the Φ evaluation function and provide more insights into its real working. That is why this analysis 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 Subsection 3.1 was coded in C, named SD-LA and SD-Φ depending on which evaluation function is used. The algorithm was compiled with *gcc* using the optimization flag -$O3$, and ran sequentially into a cluster of 10 nodes, each having a Xeon bi-CPU at 2 GHz, 1 GB of RAM and Linux.

The test-suite used in this experiment is composed of the 21 benchmarks[2] proposed in [12] and used later in [10, 13, 14]. In our preliminary experiments, we observed similar results over the set of 21 benchmark instances. For the reason of space limitation, we have decided to report only the results of 10 representative instances covering the different cases.

The methodology used consistently throughout this experimentation is the following. First, 20 random arrangements were generated for each of the 10 selected benchmark instances, and were used as starting solutions for each run of the compared evaluation functions. The average results achieved in these executions are summarized in Table 2, where column 1 and 2 show the name of

[2] http://www.lsi.upc.es/~jpetit/MinLA/Experiments

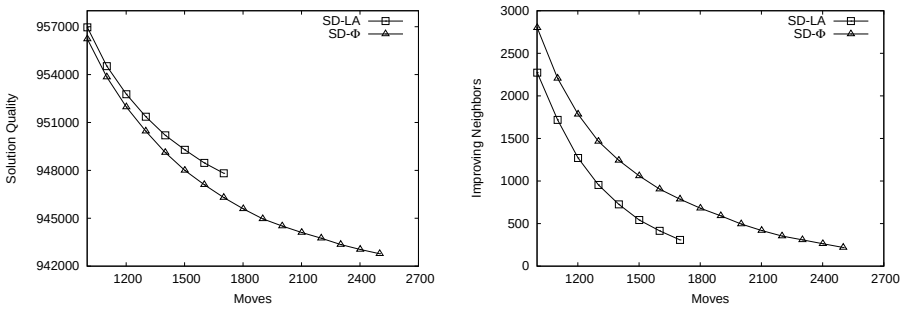


Fig. 2. Graphs representing the behavior of the compared evaluation functions over the *randomA*1 instance. (a) Average solution quality, (b) Average improving neighbors.

the graph and its number of vertices. Columns 2 to 8 display the total iterations (I), the final cost in terms of total edge length (C), and the CPU time per iteration (T) in seconds for both SD-LA and SD-Φ respectively. Column 9 presents the average number of improving neighbors (IN) found by SD-Φ, at the same iteration where SD-LA stops, that is when IN for SD-LA equals zero. Last column shows the difference (Δ_C) between the total average cost produced by the compared algorithms.

The results presented in Table 2 show clearly that the SD algorithm that employs Φ, consistently has better results than the algorithm that uses LA. The average improvement obtained with the use of Φ is -50029.6, which leads to a significant decrease of the total edge length (Δ_C up to -237163.3). Notice that the SD-LA algorithm always stops the search process earlier than SD-Φ (compare columns 2 and 5), basically because LA can not distinguish arrangements with the same total edge length given as consequence a critical deficiency in finding improving neighbors (see column 8). Additionally, it is important to remark that these results can be obtained without a significant increment in the computing time, one iteration of SD-LA is approximatively equal to one iteration of SD-Φ (see columns 4 and 7).

The dominance of Φ is better illustrated in Fig. 2, where the behavior of the studied evaluation functions is presented over the *randomA*1 instance (the rest of the studied instances provide similar results). In Fig. 2(a) the X axis represents the number of moves, while the Y axis indicates the average solution quality. Fig. 2(b) depicts the evolution of the average number of improving neighbors (Y axis) with respect to the number of moves. Observe that SD-Φ produces better results because it is capable of identifying the improving neighbors that orient better the search process.

4 Comparing the Evaluation Functions within a Memetic Algorithm

After having studied the characteristics of Φ by using a simple SD algorithm, we have decided to evaluate its practical usefulness within a Memetic Algorithm (MA).

4.1 Memetic Algorithm

The MA implementation used for this comparison was kept as simple as possible to obtain a clear idea of the evaluation function effectiveness. Indeed, the recombination operator does not take into account the individuals' semantic, no special initialization procedure is employed. Furthermore, A simple SD algorithm was used, which is not as effective as Simulated Annealing or Tabu Search, to reduce the strong influence of (sophisticated) local search procedures. In this sense, this MA implementation is much simplified comparing with the MA of [13]. Next all the details of our MA implementation are presented.

Search Space, Representation and Fitness Function. The search space, representation and evaluation (fitness) functions are the same used in the SD algorithm presented in Section 3.1.

Initialization. The population P is initialized with $|P|$ configurations randomly generated.

Selection. In this MA mating selection is performed by tournament selection, while selection for survival is done by choosing the best individuals from the pool of parents and children, taking care that each phenotype exists only once in the new population (a $(\mu + \lambda)$ selection scheme).

Recombination Operator. For this implementation the Partially Matched Crossover (PMX) operator, introduced in [5], was selected. PMX is designed to preserve absolute positions from both parents.

Local Search Operator. Its purpose is to improve the configurations produced by the recombination operator. In this MA we have decided to use a modified version of the SD algorithm presented in Section 3.1. Instead of replacing the current solution with the best arrangement found in its neighborhood, it is replaced with the first improving neighbor. Notice that this Descent Algorithm is weaker than its best improvement version used in Section 3.1, but it is faster. This process is repeated until no better arrangement is found or the predefined maximum number iterations is reached.

General Procedure. MA starts building an initial population P. Then at each generation, a predefined number of recombinations (*offspring*) are executed. In each recombination two configurations are chosen by tournament selection from the population, then a recombination operator is used to produce two offspring. The local search operator is applied to improve both offspring for a fixed number of iterations L and the improved configurations are inserted into the population. Finally, the population is updated by choosing the best individuals from the pool of parents and children. This process repeats until a predefined number of generations ($maxGenerations$) is reached.

Table 3. Performance comparison between MA-LA and MA-Φ.

	MA-LA			MA-Φ			
Graph	C	Dev.	T	C	Dev.	T	Δ_C
randomA1	882305.9	3833.1	937.7	877033.1	4371.4	930.2	-5272.8
randomA3	14243888.8	11272.2	7843.6	14235917.8	12236.3	9030.9	-7971.0
bintree10	13869.0	5274.3	171.2	13422.3	4119.3	176.7	-446.8
mesh33x33	35151.8	359.3	65.7	35083.4	90.0	129.9	-68.3
3elt	458695.8	7230.1	10968.2	453149.4	5176.7	11822.6	-5546.4
airfoil1	382781.0	2701.0	8855.5	380404.6	4464.6	9515.3	-2376.5
c2y	85240.5	733.6	370.4	84201.9	723.2	405.8	-1038.6
c3y	137983.9	1027.1	686.7	137281.7	1746.9	705.9	-702.2
gd95c	506.2	0.6	0.4	506.1	0.4	0.5	-0.1
gd96a	102827.8	1759.9	319.9	102285.3	2052.9	348.1	-542.4
			Average				-2396.5

4.2 Computational Experiments

For this comparison the MA presented in Section 4.1 was coded in C. Let us call it MA-LA or MA-Φ to distinguish which evaluation function it employs. The algorithm was compiled with *gcc* using the optimization flag -$O3$ and run in the computational platform described in Subsection 3.2. The same parameters were used for MA-LA and MA-Φ in this comparison: a) population size $|P| = 50$, b) recombinations per generation *offspring*$= 5$, c) maximal number of local search iterations $L = 0.20 * |V|$ and d) maximal number of generations $maxGenerations = 1000$.

Table 3 presents the average results obtained in 20 independent executions for each of the 10 benchmark instances selected for the experiments of the Subsection 3.2. The first column in the table shows the name of the graph. The rest of the columns indicate the cost in terms of total edge length (C), its standard deviation (Dev.) and the total CPU time (T) in seconds for the MA-LA and MA-Φ algorithms respectively. Finally, column 8 displays the difference (Δ_C) between the cost found by MA-Φ and that reached by MA-LA.

From Table 3, one observes that MA-Φ is able to improve on the 10 selected instances the results produced by MA-LA. With respect to the computational effort we have noted that MA-Φ consumes approximately the same computing time than MA-LA. So this second experiment confirms again that Φ is superior than LA.

4.3 Using Φ within a more Sophisticated MA

Given the results obtained with the simple MA described in the Subsection 4.1, we have decided to asses the performance of Φ within a more sophisticated MA. For this purpose we have reused the MA reported in [13] and replace in its code the classic evaluation function by the refined function Φ (call this algorithm MAMP-Φ). The resulting code was compiled in the computational platform described in 3.2 and executed 20 times on the full test-suite of Petit [12] using the parameters suggested in [13].

The results of this experiment are presented in Table 4 and compared with those of the two best known heuristics: AMG [14] and MAMP [13]. In this table,

Table 4. Performance comparison between MAMP-Φ and the state-of-the-art algorithms.

				MAMP-Φ				
Graph	$\|V\|$	AMG	MAMP	C	Avg.	Desv.	T	Δ_C
randomA1	1000	888381	867535	867214	867581.7	634.2	909.0	-321
randomA2	1000	6596081	6533999	6532341	6534770.7	2616.2	3486.3	-1658
randomA3	1000	14303980	14240067	14238712	14239766.4	1213.0	5175.7	-1355
randomA4	1000	1747822	1719906	1718746	1720260.5	1479.7	1904.1	-1160
randomG4	1000	140211	141538	140211	140420.7	354.1	2077.2	0
bintree10	1023	3696	3790	3721	3749.7	36.3	978.7	25
hc10	1024	523776	523776	523776	523776.2	0.6	1140.8	0
mesh33x33	1089	31729	31917	31789	31847.5	68.8	1123.2	60
3elt	4720	357329	362209	357329	361174.7	2198.0	5609.9	0
airfoil1	4253	272931	285429	273090	278259.8	7063.0	5443.1	159
whitaker3	9800	1144476	1167089	1148652	1160117.7	9207.0	15299.7	4176
c1y	828	62262	62333	62262	62302.7	65.0	643.5	0
c2y	980	78822	79017	78929	78964.2	45.4	654.8	107
c3y	1327	123514	123521	123376	123458.5	92.3	728.1	-138
c4y	1366	115131	115144	115051	115129.1	185.7	733.3	-80
c5y	1202	96899	96952	96878	97080.5	391.9	715.3	-21
gd95c	62	506	506	506	506.1	0.3	1.6	0
gd96a	1096	96249	96253	95242	96019.4	559.9	636.8	-1007
gd96b	111	1416	1416	1416	1416.1	0.3	3.7	0
gd96c	65	519	519	519	519.7	1.3	1.4	0
gd96d	180	2391	2391	2391	2391.5	1.1	7.7	0

the name of the graph and its number of vertices are displayed in the first two columns. The best solution reported by AMG and MAMP is shown in columns 3 and 4 respectively. Columns 5 to 8 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 for the MAMP-Φ algorithm. Last column shows the difference (Δ_C) between the best cost produced by MAMP-Φ and the previous best-known solution.

From Table 4, one observes that MAMP-Φ is able to improve on 8 previous best known solutions and to equal these results in 8 more instances. In 5 instances, MAMP-Φ did not reach the best reported solution, but its results are very close to them (in average 0.028%). Moreover, MAMP-Φ improves in 16 instances the results achieved by MAMP (see columns 4 and 5).

5 Conclusions

In this paper, we have introduced the Φ evaluation function for the Minimum Linear Arrangement problem. It allows to indicate the potential for further improvement of an arrangement by considering additional information induced by the absolute differences between adjacent labels of the arrangement. To gain more insights into its real working, the classical evaluation function LA and Φ were compared over a set of well known benchmarks within a basic SD algorithm. The results showed that an average improvement of 3.39% can be achieved when Φ is used, because it is able to identify an average number of improving neighbors greater than that produced by LA.

Moreover, we have evaluated the practical usefulness of Φ within a basic MA. From this experiment, it is observed that MA-Φ is able to improve on the 10

selected instances the results produced by MA-LA, consuming approximately the same computing time.

Finally, in a third experiment the Φ evaluation function was incorporated into a more sophisticated MA. The resulting algorithm, called MAMP-Φ, was compared with the two best known heuristics: AMG [14] and MAMP [13]. The results obtained by MAMP-Φ are superior to those presented by the previous proposed evolutionary approach [13], and permit to improve on 8 previous best known solutions.

This study confirms that the research on evaluations functions, to provide more effective guidance for heuristic algorithms, is certainly a very interesting way to boost the performance of these algorithms.

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. I. Charon and O. Hudry. The noising method: A new method for combinatorial optimization. *Operations Research Letters*, 14(3):133–137, 1993.
2. J. Diaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.
3. E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.
4. M. Garey and D. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
5. D. E. Goldberg and R. Lingle. Alleles, loci, and the travelling salesman problem. In *Proc. of ICGA'85*, pages 154–159. Carnegie Mellon publishers, 1985.
6. 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*, 24:728–735, 1994.
7. L. Harper. Optimal assignment of numbers to vertices. *Journal of SIAM*, 12(1):131–135, 1964.
8. D. Johnson, C. Aragon, L. 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.
9. S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proc. Of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 819–830. IEEE Press, 1994.
10. Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. *Lecture Notes in Computer Science*, 2573:293–306, 2002.
11. Y. Lai and K. Williams. A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *Graph Theory*, 31:75–94, 1999.
12. J. Petit. *Layout Problems.* PhD thesis, Universitat Politécnica de Catalunya, 2001.
13. 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.
14. I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 2004. in press.