Memetic Algorithms 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 presents a new Memetic Algorithm designed to compute near optimal solutions for the MinLA problem. It incorporates a highly specialized crossover operator, a fast MinLA heuristic used to create the initial population and a local search operator based on a fine tuned Simulated Annealing algorithm. Its performance is investigated through extensive experimentation over well known benchmarks and compared with other state-of-the-art algorithms.

Key words: Memetic Algorithms, Linear Arrangement, Heuristics.

1 Introduction

Evolutionary algorithms (EAs), as general purpose optimization procedures, have been successfully applied in a broad spectrum of areas in physics, chemistry, engineering, management science, biology and computer science [22].

It is well recognized that it is essential to incorporate some form of domain knowledge into EAs to arrive at highly effective search [1, 4, 10]. There are many ways to achieve this, for example by the combination of EAs with other efficient problem-dependent heuristics, or by using encodings and genetic operators that are tailored to the problem to be solved. Memetic algorithms (MAs) follow such an approach and have demonstrated recently to be very efficient [3, 7, 8, 12, 16, 23]. Under different contexts and situations, MAs are also known as hybrid EAs or genetic local searchers.

In this paper, we are interested in tackling with the use of MAs a well-known combinatorial optimization problem: the Minimum Linear Arrangement problem (MinLA). Garey and Johnson have shown that finding the minimum linear arrangement of a graph is NP-hard and the corresponding decision problem is NP-complete [9]. MinLA was first stated by Harper [11]. His aim was to design error-correcting codes with minimal average absolute errors on certain classes

* 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).

of graphs. The MinLA problem arises also in other application areas like graph drawing, VLSI layout, software diagram layout and job scheduling [5].

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

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

Then the MinLA problem consists in finding an arrangement φ for a given G so that $LA(G, \varphi)$ is minimized.

There exist polynomial time exact algorithms for some special cases of MinLA such as trees, rooted trees, hypercubes, meshes, outerplanar graphs, and others (see [5] for a detailed survey). However, MinLA is NP-hard for general graphs [9] and for bipartite graphs [6]. 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 balanced decomposition tree heuristic (DT) [2], the multi-scale algorithm (MS) [14] and the algebraic multigrid scheme (AMG) [21]; and b) metaheuristics such as Simulated Annealing [17–19] and Genetic Algorithms [20].

This paper aims at developing a powerful Memetic Algorithm (MA) for finding near optimum solutions for the MinLA problem. To achieve this, the new algorithm, called MAMP (standing for Memetic Algorithm for the MinLA Problem), incorporates a highly specialized crossover operator, a fast MinLA heuristic used to create the initial population and a local search operator based on a fine tuned Simulated Annealing algorithm. The performance of MAMP is assessed with a set of 21 benchmark instances taken from the literature. The computational results are reported and compared with previously published ones, showing that our algorithm is able to improve on some previous best results.

The paper is organized as follows. Section 2 reviews some existing solution procedures for the MinLA problem. Then, the different components of our MA are presented in Section 3. Section 4 is dedicated to computational experiments and comparisons with previous results. The last section summarizes the main contributions of this research work.

2 Relevant Existing Procedures

Because of the importance of the MinLA problem, much research has been carried out in developing effective heuristics for it. In this section, we give a brief review of three representative algorithms which were used in our comparisons.

2.1 The SS+SA Heuristic

In 2001 Jordi Petit developed a heuristic for the MinLA problem, called SS+SA [18, 19]. It works as follows: First a global solution is obtained by using Spectral

Sequencing (SS), a method originally proposed by Juvan and Mohar, which is based on the computation of the Fiedler vector of G [13]. Then the resulting arrangement is iteratively improved using a SA algorithm previously reported in [17]. It performs local changes based on a special neighborhood distribution, called FlipN, that tends to favor moves with high probability to be accepted.

The SS+SA algorithm proposed by Petit starts at an initial temperature $T_0 = 10$, at each Metropolis round $r = 20n^{3/2}$ moves are generated. Then the current temperature is decremented with the relation $T_k = \alpha T_{k-1}$, with $\alpha = 0.95$ until to reach a final temperature $T_f = 0.2$. The author claims that these parameters were fixed based on some preliminary experiments.

The author makes a computational comparison of the SS procedure, a SA algorithm and the combination of both methods (SS+SA). For this comparison Petit collected a set of 21 benchmark graphs. The test-suite consists of 5 random graphs, 3 "regular"graphs (a hypercube, a mesh, and a binary tree), 3 graphs from finite element discretizations, 5 graphs from VLSI designs, and 5 graphs from graph drawing competitions.

The experiments have shown that for the finite element discretization graphs SS+SA improves the SS and SA solutions by more than 20%, while reducing the running time to a 25% of SA. For the rest of the graphs, SS+SA allways improves the SS solutions and only for two graphs ($c5y$ and $gd96a$) it is unable to improve the SA solution. The running times are usually lower for SS+SA than for SA. The author concludes that the SS+SA heuristic is a valuable improvement over the SS and SA methods.

2.2 The DT+SA Heuristic

Besides Petit's work, Bar-Yehuda *et al.* present in [2] a divide-and-conquer approach to the MinLA problem. Their idea is to divide the vertices into two sets, to recursively arrange each set internally at consecutive locations, and finally to join the two ordered sets, deciding which will be put to the left of the other.

The computed arrangement is specified by a decomposition tree (DT) that describes the recursive partitioning of the subproblems. Each vertex of the tree gives a degree of freedom as to the order in which the two vertex sets are glued together. Thus, the goal of the algorithm is to decide for each vertex of the decomposition tree the order of its two children. The authors propose a dynamic programming algorithm for computing the best possible ordering for a given decomposition tree.

The set of benchmark instances used in [2] is the same proposed by Petit in [17–19]. They applied their algorithm iteratively, starting each iteration with the result of the previous one. After a few tens of iterations, the algorithm usually yields results within 5-10% of those obtained by Petit's SA, but at a fraction of its running time. They have used these computed arrangements as an initial solution for the SA reported in [17] and slightly better results were obtained.

2.3 The MS Heuristic

In 2002, Koren and Harel present a linear-time algorithm for the MinLA problem, based on the multi-scale (MS) paradigm [14]. MS techniques transform a high-dimensional problem in an iterative fashion into subproblems of increasingly lower dimensions, via a process called coarsening. On the coarsest scale the problem is solved exactly, following which a refinement process starts, whereby the solution is progressively projected back into higher and higher dimensions, updated appropriately at each scale, until the original problem is reproduced and solved.

The algorithm proposed in [14], starts with a preprocessing stage that obtains, rapidly, a reasonable linear arrangement by using spectral sequencing and then improves the result by applying a procedure, that they call *median iteration*, for about 50 sweeps. The median iteration is a randomized algorithm based on a continuous relaxation of the MinLA problem, where vertices are allowed to share the same place, or to be placed on non-integral points.

Then, the MS algorithm starts by refining the arrangement locally. The intention of the refinement is not only to minimize the arrangement cost, but also to improve the quality of the coarsening step that follows. The next step is to coarsen the graph based on restricting consecutive vertex pairs of the current arrangement. The problem is then solved in the restricted solution space, by running all this set of steps (called a *V-cycle*) recursively on the coarse graph. Once a good solution is found in the restricted solution space, the algorithm refines it locally (in the full solution space).

Koren and Harel have also used the set of test instances proposed by Petit. For each graph in this set, they ran their MS algorithm first with a single V-cycle and then with ten. They present these results as well as those obtained during the preprocessing stage (spectral sequencing and median iteration algorithms). The quality of their results after 10 V-cycle iterations is comparable to that of Petit's SA, but the running time is significantly better.

Later in 2004, an improvement to the algorithm proposed by Koren and Harel was presented in [21]. The main difference between these approaches is the coarsening scheme. Koren and Harel use *strict* aggregation, while Safro *et al.* use *weighted* aggregation. In a strict aggregation procedure the nodes of the graph are blocked into small disjoint subsets, called aggregates. By contrast, in the weighted aggregation each node can be divided into fractions, and different fractions belong to different aggregates. Safro *et al.* have shown experimentally that their approach can obtain high quality results in linear time for the MinLA problem and can be considered as one of the best MinLA algorithms known today.

2.4 The Genetic Hillclimbing Algorithm

In [20] a Genetic Hillclimbing (GH) algorithm is proposed. It represents linear arrangements as permutations of vertices and operates as follows: An initial population $|P| = 100$ is created by combining one individual generated with spectral

sequencing, 10% of randomly generated individuals and the rest is generated using depth-first and breadth-first search algorithms initialized with a randomly chosen vertex. At each generation $0.5|P|$ pairs of individuals are randomly selected, then a two point crosssover with unfeasibility repair is applied with 98% of probability in order to produce two offspring each time. Both resulting offspring are compared with their parents. If offspring has better fitness than one of its parents, then it is inserted in the population else the parent is taken back to the population and the offspring is eliminated. After that, $nlog(n)$ hillclimbing steps are applied to each individual of the population. It allows to obtain locally optimal solutions that will be mutated with probability 15%. The mutation operator consists in applying one random swap. The process is repeated until the number of 20000 generations is reached or when 100 successive generations do not produce a better solution.

For his comparisons the author employs the set of benchmark instances proposed by Petit [17–19]. Their results show that GH has found slightly better results for 7 instances (over 21 graphs).

3 A New Memetic Algorithm for MinLA

In this section we present a new Memetic algorithm, called MAMP, for solving the MinLA problem. Next all the details of its implementation are presented.

3.1 Search Space, Representation and Fitness Function

The search space $\mathcal{A}$ for the MinLA problem is composed of all possible arrangements from V to $\{1, 2, ..., n\}$. It is easy to see then, that there are $n!$ possible linear arrangements for a graph with n vertices.

In our MA 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 φ is evaluated by using Equation 1.

3.2 The General Procedure

MAMP starts building an initial population P, which is a set of configurations having a fixed constant size $|P|$ (*initPopulation*). Then it performs a series of cycles called generations. At each generation, a predefined number of recombinations (*offspring*) are executed. In each recombination two configurations a and b are chosen randomly from the population (*selectParents*). A recombination operator is then used to produce an offspring c from a and b (*recombineIndividuals*). The local search operator (*localSearch*) is applied to improve c for a fixed number of iterations L and the improved configuration c is inserted in the population. Finally, the population is updated by choosing the best individuals from the pool of parents and children (*UpdatePopulation*). This process repeats until a stop condition is verified, usually when a predefined number of generations (*maxGenerations*) is reached. Note however, that the algorithm may stop before

reaching $maxGenerations$, if a better solution is not produced in a predefined number of successive generations ($maxFails$).

3.3 The Initialization Operator

The operator $initPopulation(|P|)$ initiates the population P with $|P|$ configurations. To create a configuration, we use the greedy frontal increase minimization (FIM) algorithm of McAllister [15], slightly adapted in order to work in a randomized form. The algorithm is based on the following two basic steps: 1) Select a starting vertex and place it in position 1. 2) For each remaining position 2 through n, select one of the unplaced vertices for placement in the current position by using the FIM strategy. It consist in selecting for placement i a vertex that is adjacent to the fewest vertices in $U_i - F_i$, where $F_i = \{u \in U_i | v \in P_i \text{ and } (u, v) \in E\}$ denotes the *front* at placement i, P_i represents the set of $i - 1$ vertices placed so far and U_i the set of currently unplaced vertices.

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

Due to the randomness of the greedy algorithm, the configurations in the initial population are quite different. This point is important for population based algorithms because a homogeneous population cannot efficiently evolve.

3.4 Selection

Mating selection ($selectParents(P)$) prior to recombination is performed on a purely random basis without bias to fitter individuals, while selection for survival ($UpdatePopulation(P)$) is done by choosing the best individuals from the pool of parents and children. It is done by taking care that each phenotype exists only once in the new population. Thus, replacement in our algorithm is similar to the (μ, λ) selection scheme used in [16].

3.5 The Recombination Operator

The main idea of the recombination operator ($recombineIndividuals(a, b)$) is to generate diversified and potentially promising individuals. To do that, a good MinLA recombination operator should take into consideration, as much as possible, the individuals' semantic.

In this subsection we present a new recombination operator LGX (*local greedy crossover*) that is able to preserve certain information contained in both parents, while some subgraphs are locally improved using a greedy mechanism. The new LGX operator works in four basic steps:

First, all the labels found at the same vertex in the two parents are assigned to the corresponding vertex in the offspring. Next, for each labeled vertex in the offspring a greedy mechanism is applied to find the labels for its adjacent vertices; this procedure tends to minimize the local MinLA contribution of each of these subgraphs. Then, for each unlabeled vertex in the offspring we take, if possible, the label from the same vertex of one of the parents. Finally, the labels for the remaining vertices are randomly assigned. The functioning of the LGX operator is presented in Algorithm 1.

```
recombineIndividuals(a, b)
begin
    // The number of assigned labels in the offspring
    assigned = copyIdenticLabels(a, b, c);
    for each vertex i labeled in c do
        assigned += localGreedy(i, c);
    end
    assigned += completeFromParents(a, b, c);
    if assigned < |V| then
        completeRandom(c, assigned);
    end
    return The offspring c;
end
```

Algorithm 1: The LGX recombination operator

3.6 The Local Search Operator

The purpose of the local search (LS) operator $localSearch(c, L)$ is to improve a configuration c produced by the recombination operator for a maximum of L iterations before inserting it into the population. In general, any local search method can be used. In our implementation, we have decided to use Simulated Annealing (SA).

In our SA-based LS operator 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 flipping the labels of any pair of different vertices from φ. We call this flipping operation a *move*. Besides the apparent simplicity of this neighborhood function, the reasons to choose it are: the easiness to perform movements and the low effort necessary to compute incrementally the cost of the new arrangement.

The SA operator starts at an initial temperature $T_0 = 10$, at each Metropolis round $r = 1000$ moves are generated. If the cost of the attempted move decreases then it is accepted. Otherwise, it is accepted with probability $P(\Delta) = e^{-\Delta/T}$

where T is the current temperature and Δ is the increase in cost that would result from that particular move. At the end of each Metropolis round then the current temperature is decremented by a factor of $\alpha = 0.955$. The algorithm stops either if the current temperature reaches $T_f = 0.001$, or when it reaches the predefined maximum of L iterations.

The algorithm memorizes and returns the most recent arrangement φ^* among the best configurations found: after each accepted move, the current configuration φ replaces φ^* if $LA(G,\varphi) \leq LA(G,\varphi^*)$ (and not only if $LA(G,\varphi) < LA(G,\varphi^*)$). The rational to return the last best configuration is that we want to produce a solution which is as far away as possible from the initial solution in order to better preserve the diversity in the population.

4 Computational Experiments

In this section, we present a set of experiments accomplished to evaluate the performance of the MA algorithm presented in Section 3. The algorithms were coded in C and compiled with *gcc* using the optimization flag -*O*3. They were run sequentially into a cluster of 10 nodes, each having a Xeon bi-CPU at 2 GHz, 1 GB of RAM and Linux. Due to the non-deterministic nature of the algorithms, 20 independent runs were executed for each of the selected benchmark instances. When averaged results are reported, they are based on these 20 corresponding runs.

In all the experiments the following parameters were used for MAMP: a) population size $|P| = 40$, b) recombinations per generation $offspring = 4$, c) maximal number of local search iterations $L = 150000$, d) maximal number of generations $maxGenerations = 10000$ and e) maximal number of successive failed generations $maxFails = 100$.

4.1 Benchmark Instances and Comparison Criteria

The test-suite that we have used in the experiments is the same proposed by Petit [17] and used later in [2, 14, 20, 21]. It consists of six different families of graphs: Uniform random (randomA* class), geometric random (randomG* class), graphs with known optima (trees, hypercubes and meshes), finite element discretizations (3elt, airfoil1 and whitaker3), VLSI design (c*y class) and graph drawing competitions (gd* class). All of them have 1000 vertices or more, except for some instances in the gd* class. These instances are available at: http://www.lsi.upc.es/~jpetit/MinLA/Experiments

The criteria used for evaluating the performance of the algorithms are the same as those used in the literature: the best total edge length found for each instance and the CPU time in seconds.

4.2 Comparison Between MAMP and GH

The purpose of the first experiment is to compare our memetic algorithm MAMP with the previous one of [20] (GH). To enable a fair comparison we have obtained

the GH source code[1]. Then GH and MAMP were compiled and executed in our hardware and operating system platform 20 times on each benchmark instance.

The parameters for the GH algorithm are those reported in Poranen's work: a) population with 100 individuals, b) 50 crossovers per generation, c) 98% crossover rate, d) 15% mutation rate, e) $nlog(n)$ hillclimbing steps, f) a maximum of 20000 generations and g) at maximum 100 successive failed generations. We would like to point out that GH employs a population of 100 individuals, while MAMP has a population size of 40. We have decided to conserve this difference, apparently unfavorable for MAMP, because in a preliminary experiment we have tried to reduce the GH population size to 40, but the results produced by GH were inferior in solution quality.

Table 1. Performance comparison between MAMP and GH.

	GH			MAMP			
Graph	Bc	Avg	t	Bc	Avg	t	Δ_C
randomA1	878705	883138.2	4079.2	867535	868480.4	918.7	-11170
randomA2	6557701	6564256.4	24010.2	6533999	6536249	3477.4	-23702
randomA3	14253230	14253230	25629.1	14240067	14240757	5221.2	-13163
randomA4	1735414	1735414	10066.2	1719906	1721070.4	1904.1	-15508
randomG4	153470	153470	1924.6	141538	143855	2097.2	-11932
bintree10	3873	3920.6	413.0	3790	3812.8	984.7	-83
hc10	523776	523776	325.8	523776	523776	1152.4	0
mesh33x33	31968	32127.2	1129.9	31917	31979.8	1177.9	-51
3elt	397305	403654.2	41952.7	362209	364403	5758.9	-35096
airfoil1	300656	300656	74023.7	285429	286986.6	5542.4	-15227
whitaker3	1189831	1189831	9006538.7	1167089	1168140.25	15322.4	-22742
c1y	63063	63440.6	783.9	62333	62383.6	651.5	-730
c2y	80453	81914.2	935.4	79017	80998	672.8	-1436
c3y	129775	130789.4	2092.0	123521	123689.4	731.1	-6254
c4y	118270	119277	2796.8	115144	115406	739.4	-3126
c5y	100877	102054.8	1983.7	96952	97219.4	741.5	-3925
gd95c	506	508.4	2.2	506	506.2	1.5	0
gd96a	105947	108714.6	886.7	96253	96384.8	667.9	-9694
gd96b	1416	1417.2	4.5	1416	1416.2	3.3	0
gd96c	519	519.2	2.0	519	520	1.4	0
gd96d	2406	2413.6	10.9	2391	2392	8.1	-15
Average							-8278.8

The results obtained from comparing both algorithms are presented in Table 1. Column 1 shows the name of the graph. Columns 2 to 7 display the best cost (Bc), the average cost (Avg) and the average CPU time (t) in seconds for finding the best solution in each one of the 20 runs of the GH and MAMP algorithms respectively. Last column presents the difference (Δ_C) between the best cost found by MAMP and the best cost produced by GH.

[1] available at http://www.cs.uta.fi/~tp/optgen/index.html

Table 1 shows clearly that MAMP allows us to obtain better results for many classes of graphs with less computing time. We can observe an important improvement in cost in 17 out of 21 instances. For the rest of the instances the results of MAMP equal those produced by GH, but always with less computational effort, thanks to its reduced population size.

4.3 Comparison Between MAMP and the Best Known Results

In the second experiment a performance comparison of our MAMP procedure with the following heuristics was carried out: SS+SA [18, 19], DT+SA [2], AMG [21] and GH [20]. Table 2 presents the detailed computational results produced by this experiment. The first three columns in the table indicate the name of the graph, its number of vertices and its number of edges. The rest of the columns indicate the best total edge length found by each of the compared heuristics. These results were taken from their corresponding paper. Finally, last column presents the difference (Δ_C) between the best total edge length found by MAMP and the previous best known solution reported in the literature.

From Table 2, one observes that MAMP is competitive in terms of solution quality. MAMP is able to improve on 4 previous best known solutions and to equal these results in 5 instances. For the other instances, MAMP did not reach the best reported solution, but its results are very close to the best reported (in average 1.009%). Notice that for some instances the improvement is important; leading to a significant decrease of the total edge length (Δ_C up to -6579).

Even if the results obtained by our memetic algorithm are very competitive we observe that MAMP, given that it is a memetic algorithm, consumes considerably more computer time than some heuristics for MinLA such as DT [2], MS [14] and AMG [21].

5 Conclusions

In this paper, a MA designed to compute near optimal solutions for the MinLA problem was presented. This algorithm, called MAMP, is based on the use of a greedy vertex-by-vertex algorithm for generating the initial population of the MA, a fine tuned Simulated Annealing algorithm for finding local optima in the search space, and a highly specialized crossover operator for efficiently explore the space of local optima in order to find the global optimum.

The performance of our MAMP algorithm was assessed through extensive experimentation over a set of well known benchmark instances and compared with four other state-of-the-art algorithms: SS+SA [18, 19], DT+SA [2], AMG [21] and GH [20]. The results obtained by MAMP are superior to those presented by the previous proposed evolutionary approach [20], and permit to improve on some previous best known solutions.

There are some issues for future research. For example, to investigate the behavior of MAMP when it is applied to larger instances, like those proposed by Koren and Harel in [14], in order to study its scalability. Additionally, the

Table 2. Performance comparison between MAMP and several state-of-the-art algorithms.

Graph	$\|V\|$	$\|E\|$	SS+SA	DT+SA	AMG	GH	MAMP	Δ_C
randomA1	1000	4974	869648	884261	888381	878637	867535	-2113
randomA2	1000	24738	6536540	6576912	6596081	6550292	6533999	-2541
randomA3	1000	49820	14310861	14289214	14303980	14246646	14240067	-6579
randomA4	1000	8177	1721490	1747143	1747822	1735691	1719906	-1584
randomG4	1000	8173	150940	146996	140211	142587	141538	1327
bintree10	1023	1022	4069	3762	3696	3807	3790	94
hc10	1024	5120	523776	523776	523776	523776	523776	0
mesh33x33	1089	2112	31929	33531	31729	32040	31917	188
3elt	4720	13722	363686	363204	357329	383286	362209	4880
airfoil1	4253	12289	285597	289217	272931	306005	285429	12498
whitaker3	9800	28989	1169642	1200374	1144476	1203349	1167089	22613
c1y	828	1749	63145	62333	62262	62562	62333	71
c2y	980	2102	79429	79571	78822	79823	79017	195
c3y	1327	2844	123548	127065	123514	125654	123521	7
c4y	1366	2915	116140	115222	115131	117539	115144	13
c5y	1202	2557	97791	96956	96899	98483	96952	53
gd95c	62	144	509	506	506	506	506	0
gd96a	1096	1676	96366	99944	96249	98388	96253	4
gd96b	111	193	1416	1422	1416	1416	1416	0
gd96c	65	125	519	519	519	519	519	0
gd96d	180	228	2393	2409	2391	2391	2391	0

performance of MAMP should be more deeply investigated with other parameter settings for population size, operator rates and stopping conditions.

Acknowledgments. The authors would like to thank Andrew J. McAllister who has kindly provided us with his source code. The reviewers of the paper are greatly acknowledged for their constructive comments.

References

1. T. Bäck, U. Hammel, and H. P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.
2. R. Bar-Yehuda, G. Even, J. Feldman, and S. Naor. Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. *Journal of Graph Algorithms and Applications*, 5(4):1–27, 1996.
3. D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, editors. *New Ideas in Optimization (Part 4: Memetic Algorithms)*. McGraw-Hill, 1999.
4. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrad, New York, 1991.
5. J. Diaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

6. S. Even and Y. Shiloah. NP-completeness of several arrangement problems. Technical Report CS0043, Computer Science Department, Technion, Israel Institute of Technology, Haifa, Israel, January 1975.
7. B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 616–621. IEEE Press, 1996.
8. P. Galinier and J. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
9. M. Garey and D. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
10. J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60, London, 1987. Morgan Kaufmann Publishers.
11. L. Harper. Optimal assignment of numbers to vertices. *Journal of SIAM*, 12(1):131–135, 1964.
12. W. E. Hart, N. Krasnogor, and J. E. Smith, editors. *Recent Advances in Memetic Algorithms and Related Search Technologies*. Springer-Verlag, 2004.
13. M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992.
14. Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. In L. Kucera, editor, *Proceedings of 28th Inter. Workshop on Graph-Theoretic Concepts in Computer Science (WG'02)*, volume 2573 of *LNCS*, pages 293–306. Springer Verlag, 2002.
15. A. J. McAllister. A new heuristic algorithm for the linear arrangement problem. Technical Report TR-99-126a, Faculty of Computer Science, University of New Brunswick, 1999.
16. P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms and greedy operators for graph bi-partitioning. *Evolutionary Computation*, 8(1):61–91, 2000.
17. J. Petit. Approximation heuristics and benchmarkings for the MinLA problem. In *Alex '98 – Building Bridges Between Theory and Applications*, pages 112–128, 1998.
18. J. Petit. *Layout Problems*. PhD thesis, Universitat Politécnica de Catalunya, 2001.
19. J. Petit. Combining spectral sequencing and parallel simulated annealing for the MinLA problem. *Parallel Processing Letters*, 13(1):71–91, 2003.
20. T. Poranen. A genetic hillclimbing algorithm for the optimal linear arrangement problem. Technical report, University of Tampere, Finland, June 2002.
21. I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 2004. in press.
22. M. Tomassini. A survey of genetic algorithms. *Annual Reviews of Computational Physics*, III:87–118, 1995.
23. X. Yao, F. Wang, K. Padmanabhan, and S. Salcedo-Sanz. Hybrid evolutionary approaches to terminal assignment in communications networks. In W. E. Hart, N. Krasnogor, and J. E. Smith, editors, *Recent Advances in Memetic Algorithms and Related Search Technologies*, pages 129–160. Springer-Verlag, 2004.