An Improved Memetic Algorithm for the Antibandwidth Problem*

Eduardo Rodriguez-Tello and Luis Carlos Betancourt

CINVESTAV-Tamaulipas, Information Technology Laboratory.
Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria Tamps., MEXICO
{ertello,lbetancourt}@tamps.cinvestav.mx

Abstract. This paper presents an Improved Memetic Algorithm (IMA) designed to compute near-optimal solutions for the antibandwidth problem. It incorporates two distinguishing features: an efficient heuristic to generate a good quality initial population and a local search operator based on a Stochastic Hill Climbing algorithm. The most suitable combination of parameter values for IMA is determined by employing a tunning methodology based on Combinatorial Interaction Testing. The performance of the fine-tunned IMA algorithm is investigated through extensive experimentation over well known benchmarks and compared with an existing state-of-the-art Memetic Algorithm, showing that IMA consistently improves the previous best-known results.

Key words: Memetic Algorithms, Antibandwidth Problem, Combinatorial Interaction Testing, Parameter Tunning

1 Introduction

The *antibandwidth* problem was originally introduced as the *separation number* problem by Leung et al. in connection with the multiprocessor scheduling problem [1]. Later, it has also received the name of *dual bandwidth* [2]. This combinatorial optimization problem consists in finding a labeling for the vertices of a graph $G(V, E)$, using distinct integers $1, 2, \ldots, |V|$, so that the minimum absolute difference between labels of adjacent vertices is maximized.

There exist practical applications of the antibandwidth problem which arise in various fields. Some examples are: radio frequency assignment problem [3], channels assignment and T-coloring problems [4], obnoxious facility location problem [5] and obnoxious center problem [6, 2].

The antibandwidth problem can be formally stated 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 bijective labeling

* This research work was partially funded by the following projects: CONACyT 99276, Algoritmos para la Canonización de Covering Arrays; 51623 Fondo Mixto CONACyT y Gobierno del Estado de Tamaulipas.

function for the vertices of G, $\varphi : V \rightarrow \{1, 2, \ldots, n\}$, the antibandwidth for G with respect to the labeling φ is defined as:

$$\mathrm{AB}_{\varphi}(G) = \min\{|\varphi(i) - \varphi(j)| : (i, j) \in E\} \tag{1}$$

Then the antibandwidth problem consists in finding a labeling (solution) φ^* for which $\mathrm{AB}_{\varphi^*}(G)$ is maximized, i.e.,

$$\mathrm{AB}_{\varphi^*}(G) = \max\{\mathrm{AB}_{\varphi}(G) : \varphi \in \mathscr{L}\} \tag{2}$$

where $\mathscr{L}$ is the set of all possible labeling functions.

Leung et al. have shown that finding the maximum antibandwidth of a graph is NP-hard for general graphs [1]. Therefore, there is a need for heuristics to address this problem in reasonable time since it is unlikely that exact algorithms running in polynomial time exist for solving it in the general case.

This paper aims at developing an Improved Memetic Algorithm (IMA) for finding near-optimal solutions for the antibandwidth problem. To achieve this, the proposed IMA algorithm incorporates two distinguishing features: a fast heuristic to create a good quality initial population and a local search operator based on a Stochastic Hill Climbing algorithm. Through the use of a tunning methodology, based on Combinatorial Interaction Testing [7], the combination of both components and parameter values for IMA was determined to achieve the best trade-off between solution quality and computational effort. The performance of IMA is assessed with a test-suite, composed by 30 benchmark instances taken from the literature. The computational results are reported and compared with previously published ones, showing that our algorithm is able to consistently improve the previous best-known solutions for the selected benchmark instances.

The rest of this paper is organized as follows. In Sect. 2, a brief review is given to present some representative solution procedures for the antibandwidth problem. Then, the components of our Improved Memetic Algorithm are discussed in detail in Sect. 3. Two computational experiments are presented in Sect. 4. The first one is dedicated to determine the best parameter settings for IMA, while the second carries out a performance comparison of IMA with respect to an existing state-of-the-art Memetic Algorithm. Finally, the last section summarizes the main contributions of this work and presents some possible directions for future research.

2 Relevant Existing Procedures

Because of the theoretical and practical importance of the antibandwidth problem, much research has been carried out in developing effective heuristics for it. Most of the previous published work on the antibandwidth problem was devoted to the theoretical study of its properties for finding optimal solutions for specific cases. Polynomial time exact algorithms are known for solving some special instances of the antibandwidth problem: paths, cycles, special trees, complete and complete bipartite graphs, meshes, tori and hypercubes [8, 2, 9–13].

The work of Bansal and Srivastava [14] in an exception. They proposed a Memetic Algorithm, called MAAMP, which starts by constructing an initial population through the use of a label assignment heuristic, called LAH. It builds a level structure of a graph using a random breadth first search. Then, it randomly choses to start the labeling process either by the even or the odd levels of the structure. The vertices of the graph belonging to the same level are labeled one at a time in a greedy manner. MAAMP continues performing a series of cycles called generations. At each generation, selection for mating is done by applying a binary tournament operator in such a way that each individual in the parents population participates in exactly two tournaments. Children are generated using a unary reproduction operator that constructs a level structure of a graph by employing an intermediate breadth first search to produce a new labeling. Then, mutation based on swapping two randomly selected labels is performed over the solutions in the children population. Finally, the population is updated when each child competes with its respective parent and the best of them becomes the parent for the next generation. This process repeats until the Memetic Algorithm ceases to make progress, i.e., when a better solution is not produced in a predefined number of successive generations. The authors argued that MAAMP was able to obtain optimal solutions for standard graphs like paths, cycles, d-dimensional meshes, tori, hypercubes and complement of power graphs. However, they recognize that their algorithm did not reach the optimal antibandwidth in the case of unbalanced trees and complete binary trees. Furthermore, Bansal and Srivastava only present detailed results of their experiments for a set of 30 random connected graphs.

3 An Improved Memetic Algorithm

In this section we present the implementation details of the key components of an Improved Memetic Algorithm (IMA) for solving the antibandwidth problem. For some of these components different possibilities were analyzed (see Sect. 4.2) in order to find the combination which offers the best quality solutions at a reasonable computational effort.

3.1 Search Space, Representation and Fitness Function

Given a graph $G = (V, E)$ with vertex set V ($|V| = n$) and edge set E. The search space $\mathscr{L}$ for the antibandwidth problem is composed of all possible labelings (solutions) from V to $\{1, 2, ..., n\}$, i.e. there exist $n!/2$ possible labelings for a graph with n vertices[1].

In our IMA a labeling φ is represented as an array l of integers with length n, which is indexed by the vertices and whose i-th value $l[i]$ denotes the label assigned to the vertex i. The fitness $\mathrm{AB}_\varphi(G)$ of the labeling φ is evaluated by using (1).

[1] Because each one of the $n!$ labelings can be reversed to obtain the same antibandwidth.

3.2 General Procedure

Our IMA implementation starts building an initial population P, which is a set of configurations having a fixed constant size $|P|$. Then, it performs a series of cycles called generations. At each generation, assuming that $|P|$ is a multiple of four, the population is randomly partitioned into ($|P|$ mod 4) groups of individuals. Within each group, the two most fit individuals are chosen to become the parents in a recombination operator. The resulting offspring are mutated, then they are improved by using a local search operator for a fixed number of iterations L. Finally, the population is updated by applying a survival selection strategy.

The iterative process described above stops when a predefined maximum number of generations ($maxGenerations$) is reached. Algorithm 1 presents the pseudo code of IMA.

Algorithm 1: Improved Memetic Algorithm (IMA).

```
IMA(A graph G(V, E))
begin
    P ← initPopulation(|P|)
    while not stopCondition() do
        for i ← 1 to offspring do
            // select two parents a, b ∈ P
            (a, b) ← selectParents(P)
            c ← recombineIndividuals(a, b)
            c' ← mutation(c)
            c'' ← localSearch(c', L)
            insertIndividual(c'', P)
        end
        P ← updatePopulation(P)
    end
    return The best solution found
end
```

3.3 Initializing the Population

After comparing different heuristics for constructing labelings for the antibandwidth problem we have decided to use a variant of the heuristic LAH reported in [14].

Our labeling heuristic constructs a level structure of a graph using a breadth first search procedure exactly like LAH does. Then, the vertices are labeled one at a time following the order of this level structure and starting randomly either by the even or the odd levels. The main difference of our labeling heuristic with respect to LAH is that we do not select the next vertex to label in a greedy manner.

In our preliminary experiments we have found that a good balance between diversity and quality for the initial population is reached using a labeling built with our heuristic combined with $|P| - 1$ distinct randomly generated labelings.

3.4 Selection Mechanisms

In this implementation mating selection ($selectParents(P)$) prior to recombination is performed by tournament selection, while one of the following standard schemes is used for the survival selection ($updatePopulation(P)$): $(\mu + \lambda)$, (μ, λ) and (μ, λ) with elitism [15].

3.5 Recombination Operators

The recombination (crossover) operator plays a very important role in any Memetic Algorithm. Indeed, it is this operator that is responsible for creating potentially promising individuals. There are several crossover operators reported in the literature that can be applied to permutation problems [16–19].

In our computational experiments, described in Section 4.2, we compare the following three crossover operators in order to identify the most suitable one for the antibandwidth problem.

The *Cycle Crossover* (CX) operator [18], which preserves the information contained in both parents in the sense that all elements of the offspring are taken from one of the parents, i.e., CX does not perform any implicit mutation. The *Partially Matched Crossover* (PMX) operator [17], designed to preserve absolute positions from both parents. The Order Crossover (OX) operator [16], which is implemented to inherit the elements between two randomly selected crossover points, inclusive, from the first parent in the same order and position as they appeared in it. The remaining elements are inherited from the second parent in the order in which they appear in that parent, beginning with the first position following the second crossover point and skipping over all elements already present in the offspring.

3.6 Mutation Operator

In our IMA implementation the mutation operator was designed to introduce diversity into the population. It starts receiving a configuration (labeling) c produced by the recombination operator. Then, every label in c is exchanged with another randomly selected one with certain probability (mutation probability). Finally, these exchange operations allow to produce a new labeling c'.

3.7 Local Search Operator

The purpose of the local search (LS) operator $localSearch(c', L)$ is to improve a configuration c' produced by the mutation 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 a Stochastic Hill Climbing (SHC) algorithm because it only needs as parameter the maximum number of iterations.

In our SHC-based LS operator the neighborhood $\mathcal{N}(\varphi)$ of a configuration φ is such that for each $\varphi \in \mathscr{L}$, $\varphi' \in \mathcal{N}(\varphi)$ if and only if φ' can be obtained by exchanging the labels of any pair of vertices from φ. The main advantage of this neighborhood function is that it allows an incremental fitness evaluation of the neighboring solutions.

The LS operator starts from the current solution $c' \in \mathscr{L}$ and at each iteration randomly generates a neighboring solution $c'' \in \mathcal{N}(c')$. The current solution is replaced by this neighboring solution if the fitness of c'' improves or equals that of c'. The algorithm stops when it reaches a predefined maximum number of iterations L, and returns the best labeling found.

4 Computational Experiments

In this section two main experiments accomplished to evaluate the performance of the proposed IMA algorithm and some of its components are presented. The objective of the first experiment is to determine both a component combination, and a set of parameter values which permit IMA to attain the best trade-off between solution quality and computational effort. The purpose of the second of our experiments is to carry out a performance comparison of IMA with respect to an existing state-of-the-art Memetic Algorithm called MAAMP [14].

For these experiments IMA was coded in C and compiled with *gcc* using the optimization flag -*O*3. It was run sequentially into a CPU Xeon at 2.67 GHz, 1 GB of RAM with Linux operating system. Due to the non-deterministic nature of the algorithm, 30 independent runs were executed for each of the selected benchmark instances in each experiment.

4.1 Benchmark Instances and Comparison Criteria

The test-suite that we have used in our experiments is the same proposed by Bansal and Srivastava [14]. It consists of 30 undirected planar graphs taken from the Rome set which are employed in graph drawing competitions. All of them have a number of vertices between 50 and 100. These instances are publicly available at: `http://www.graphdrawing.org/data`.

The criteria used for evaluating the performance of the algorithms are the same as those used in the literature: the best antibandwidth found for each instance (bigger values are better) and the expended CPU time in seconds.

4.2 Components and Parameters Tunning

Optimizing parameter settings is an important task in the context of algorithm design. Different procedures have been proposed in the literature to find the most

Table 1. Input parameters of the IMA algorithm and their selected values.

$\|P\|$	Cx	$ProbCx$	$ProbMuta$	$Survival$	$ProbLS$	L
40	CX	0.70	0.00	$(\mu + \lambda)$	0.05	1000
80	PMX	0.80	0.05	(μ, λ)	0.10	5000
120	OX	0.90	0.10	(μ, λ) with elitism	0.15	10000

suitable combination of parameter values [20–22]. In this paper we employ a tunning methodology, previously reported in [23], which is based on Combinatorial Interaction Testing (CIT) [7]. We have decided to use CIT, because it allows to significantly reduce the number of tests (experiments) needed to determine the best parameter settings of an algorithm. Instead of exhaustive testing all the parameter value combinations of the algorithm, it only analyzes the interactions of t (or fewer) input parameters by creating interaction test-suites that include at least once all the t-way combinations between these parameters and their values.

Covering arrays (CAs) are combinatorial designs which are extensively used to represent those interaction test-suites. A covering array, $\mathrm{CA}(N; t, k, v)$, of size N, strength t, degree k, and order v is an $N \times k$ array on v symbols such that every $N \times t$ sub-array includes, at least once, all the ordered subsets from v symbols of size t (t-tuples) [24]. The minimum N for which a $\mathrm{CA}(N; t, k, v)$ exists is the *covering array number* and it is defined according to the following expression: $\mathrm{CAN}(t, k, v) = min\{N : \exists\, \mathrm{CA}(N; t, k, v)\}$.

CAs are used to represent an interaction test-suite as follows. In an algorithm we have k input parameters. Each of these has v values or levels. An interaction test-suite is an $N \times k$ array where each row is a test case (i.e., a covering array). Each column represents an input parameter and a value in the column is the particular configuration. This test-suite allows to cover all the t-way combinations of input parameter values at least once. Thus, the costs of tunning the algorithm can be substantially reduced by minimizing the number of test cases N in the covering array. Next, we present the details of the tunning process, based on CIT, for the particular case of our IMA algorithm.

First, we have identified $k = 7$ input parameters used for IMA: population size $|P|$, crossover operator Cx, crossover probability $ProbCx$, mutation probability $ProbMuta$, survival selection strategy $Survival$, local search probability $ProbLS$ and maximum number of local search iterations L. Based on some preliminary experiments, $v = 3$ reasonable values (shown in Table 1) were selected for each one of those input parameters.

We have constructed the smallest possible covering array $\mathrm{CA}(40; 3, 7, 3)$, shown (transposed) in Table 2, by using the Memetic Algorithm reported in [25]. This covering array can be easily mapped into an interaction test-suite by replacing each symbol from each column to its corresponding parameter value. For instance, we can map 0 in the first column (the first line in Table 2) to $|P| = 40$, 1 to $|P| = 80$ and 2 to $|P| = 120$. The resulting interaction test-suite contains, thus, 40 test cases (parameter settings) which include at least once all the 3-way combinations between IMA's input parameters and their values[2].

[2] In contrast, with an exhaustive testing which contains $3^7 = 2187$ test cases.

Table 2. Covering array CA(40; 3, 7, 3) representing an interaction test-suite for tunning IMA (transposed).

0	2	2	1	0	1	1	2	2	0	0	1	2	0	0	0	1	1	1	0	2	0	2	2	2	0	2	0	0	1	1	1	2	0	1	2	1	1	2	0
1	0	2	1	2	1	1	1	1	2	0	2	0	2	0	1	0	2	0	1	1	0	0	2	1	0	2	0	1	0	1	0	1	2	2	0	2	2	2	0
1	0	0	2	0	2	0	2	0	2	1	2	2	0	1	2	1	1	0	0	1	2	1	2	0	1	2	0	2	0	1	2	1	1	0	0	2	1	1	0
0	2	0	1	2	0	0	2	1	0	2	2	0	1	0	1	0	0	2	2	0	1	2	1	1	1	0	0	2	1	2	2	1	2	1	1	0	1	2	0
0	2	0	0	1	2	1	0	1	1	0	1	1	0	1	1	0	2	0	0	2	0	1	2	2	2	1	2	2	1	1	2	0	2	2	0	0	1	0	1
0	1	0	2	2	0	1	0	0	0	1	1	1	1	1	2	2	1	0	2	2	0	0	1	0	2	2	0	1	1	2	2	1	0	2	2	1	0	2	2
2	0	0	2	1	0	1	1	0	2	1	0	2	2	0	1	0	2	2	0	1	0	1	1	2	2	0	1	2	2	2	1	0	0	0	1	1	1	2	2

Each one of those 40 test cases was used to executed 30 times the IMA algorithm over a subset of 6 representative graphs,[3] selected from the benchmark instances described in Sect. 4.1. The data generated by these 7200 executions is summarized in Fig. 1, which depicts the average antibandwidth reached by each test case over the 6 selected graphs, as well as the average CPU time expended.

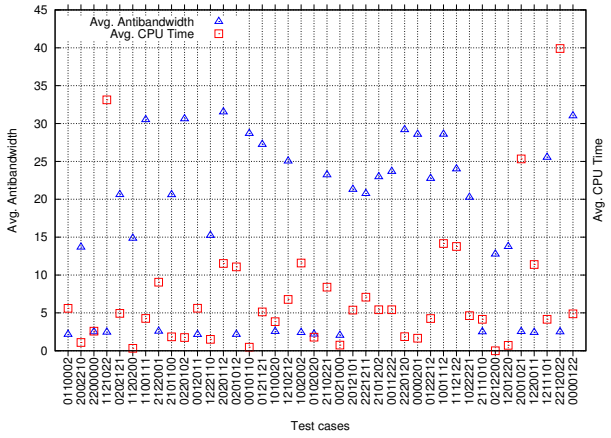

Fig. 1. Average results obtained in the tuning experiments using 40 parameter value combinations over a subset of 6 representative graphs.

From this graphic we have selected the 5 test cases which yield the best results. Their average antibandwidth and the average CPU time in seconds are presented in Table 3. This table allowed us to observe that the parameter setting giving the best trade-off between solution quality and computational effort corresponds to the test case number 40 (shown in bold). The best average antibandwidth with an acceptable speed is reached with the following input parameter values: population size $|P| = 40$, Cycle Crossover (CX) operator, crossover probability $ProbCx = 0.90$, mutation probability $ProbMuta = 0.00$, (μ, λ) survival selection strategy, local search probability $ProbLS = 0.15$ and maximum number of local search iterations $L = 10000$. These values are thus used in the experimentation reported in the next section.

[3] One graph for each size $50 \leq |V| \leq 100$ in the original set.

Table 3. Results from the 5 best parameter test cases in the tuning experiments.

Num.	Test case	Avg. antibandwidth	Avg. CPU time
13	2020112	31.527	11.802
40	**0000122**	**31.011**	**5.209**
10	0220102	30.616	2.112
7	1100111	30.511	4.611
27	2220120	29.183	2.223

4.3 Comparison Between IMA and MAAMP

In this experiment a performance comparison of the best bounds achieved by IMA with respect to those produced by the MAAMP algorithm [14] was carried out over the test-suite described in Sect. 4.1.

Table 4 displays the detailed computational results produced by this experiment. The first three columns in the table indicate the name of the graph as well as its number of vertices and edges. The theoretical upper bound (C^*) reported in [14] for those graphs is presented in Column 4. The best (C) and average ($Avg.$) antibandwidth attained by MAAMP in 10 executions and its average CPU time in seconds are listed in columns 5 to 7. These results were taken directly from [14], where a Pentium 4 at 3.2 GHz and 1 GB of RAM system was used to execute the algorithm. Next four columns provide the best (C), average ($Avg.$) and standard deviation ($Dev.$) of the antibandwidth found by IMA over 30 independent executions and the average CPU time (T) in seconds expended. The running times from MAAMP and IMA cannot be directly compared because they were executed on different computational platforms. Nevertheless, we have scaled, by a factor of 2.71, our execution times according to the Standard Performance Evaluation Corporation[4] in order to present them in a normalized form in Column 12 ($\overline{T}$). Finally, the difference (Δ_C) between the best result produced by our IMA algorithm and that achieved by MAAMP is depicted in the last column.

Analyzing the data presented in Table 4 lead us to the following main observations. First, the solution quality attained by the proposed IMA algorithm is very competitive with respect to that produced by the existing Memetic Algorithm called MAAMP [14], since IMA provides solutions whose costs (antibandwidth) are closer to the theoretical upper bounds (compare Columns 4, 5 and 8). Indeed, IMA consistently improves the best antibandwidth found by MAAMP, obtaining an average amelioration of $\Delta_C = -4.93$.

Second, one observes that for the selected instances the antibandwidth found by IMA presents a relatively small standard deviation (see Column $Dev.$). It is an indicator of the algorithm's precision and robustness since it shows that in average the performance of IMA does not present important fluctuations.

Third, we can notice that MAAMP is the most time-consuming algorithm. It uses an average of 257.79 seconds for solving the 30 selected instances, while IMA employs only 25.82 seconds (see column $\overline{T}$).

[4] `http://www.spec.org`

Table 4. Performance comparison between MAAMP and IMA over 30 undirected planar graphs from the Rome set.

				MAAMP			IMA					
Graph	$\|V\|$	$\|E\|$	C^*	C	$Avg.$	T	C	$Avg.$	$Dev.$	T	$\overline{T}$	Δ_C
ug1-50		64		18	17.40	65.28	21	20.97	0.03	8.51	23.07	-3
ug2-50		66		18	16.80	33.51	21	20.00	0.21	2.38	6.46	-3
ug3-50	50	63	24	17	16.80	52.04	21	21.00	0.00	3.61	9.77	-4
ug4-50		70		19	18.40	59.55	21	20.37	0.52	2.74	7.42	-2
ug5-50		62		20	19.80	83.44	22	21.40	0.25	12.33	33.41	-2
ug1-60		79		21	20.20	155.98	25	24.13	0.12	3.59	9.72	-4
ug2-60		81		23	21.80	21.80	24	23.53	0.26	2.41	6.54	-1
ug3-60	60	84	29	22	20.20	116.12	24	23.33	0.30	7.27	19.71	-2
ug4-60		79		22	21.20	235.20	25	24.27	0.20	6.32	17.12	-3
ug5-60		80		21	20.80	70.20	25	23.87	0.19	3.26	8.84	-4
ug1-70		98		26	23.80	354.52	28	26.83	0.35	6.13	16.60	-2
ug2-70		82		28	15.80	54.20	32	31.30	0.22	9.68	26.22	-4
ug3-70	70	82	34	27	26.20	116.31	31	30.20	0.17	5.72	15.49	-4
ug4-70		97		23	22.60	74.50	29	28.07	0.06	15.42	41.77	-6
ug5-70		88		26	25.50	323.45	29	28.47	0.26	13.03	35.31	-3
ug1-80		92		34	33.60	61.07	37	36.13	0.26	9.31	25.24	-3
ug2-80		93		31	30.20	226.90	35	34.07	0.34	14.26	38.64	-4
ug3-80	80	95	39	28	27.20	138.68	35	34.23	0.46	9.72	26.35	-7
ug4-80		101		27	25.20	582.50	34	32.80	0.23	11.05	29.95	-7
ug5-80		94		27	25.80	190.52	34	33.10	0.51	10.74	29.11	-7
ug1-90		102		32	29.80	150.40	39	38.27	0.41	8.78	23.78	-7
ug2-90		114		35	34.60	469.62	38	36.73	0.34	7.90	21.41	-3
ug3-90	90	108	44	31	29.80	316.29	40	38.83	0.49	16.84	45.64	-9
ug4-90		99		34	33.30	381.20	40	39.23	0.25	11.77	31.90	-6
ug5-90		104		34	31.80	815.47	39	38.50	0.47	10.50	28.46	-5
ug1-100		114		34	32.20	499.20	44	43.07	0.34	17.61	47.71	-10
ug2-100		114		40	38.40	715.90	44	42.80	0.23	11.22	30.40	-4
ug3-100	100	116	49	33	31.60	256.40	43	42.33	0.57	15.10	40.92	-10
ug4-100		122		35	33.80	419.80	43	42.13	0.33	15.23	41.27	-8
ug5-100		125		31	30.60	693.65	42	40.60	1.14	13.41	36.33	-11
	Avg.			27.23		257.79	32.17		0.32	9.53	25.82	-4.93

The outstanding results achieved by IMA are better illustrated in Fig. 2. The plot represents the studied instances (ordinate) against the best solution (antibandwidth) attained by the compared algorithms (abscissa). The theoretical upper bounds for these graphs are shown with squares, the previous best-known solutions provided by MAAMP [14] are depicted as circles, while the bounds computed with our IMA algorithm are shown as triangles. From this figure it can be seen that IMA consistently outperforms MAAMP, achieving for certain instances, like the graph *ug5-100*, an important increase in solution cost (Δ_C up to -11).

Thus, as this experiment confirms, our IMA algorithm is more effective than the existing Memetic Algorithm, called MAAMP.

5 Conclusions and Further Work

In this paper, an Improved Memetic Algorithm (IMA) designed to compute near-optimal solutions for the antibandwidth problem was presented. IMA's components and parameter values were carefully determined, through the use of a

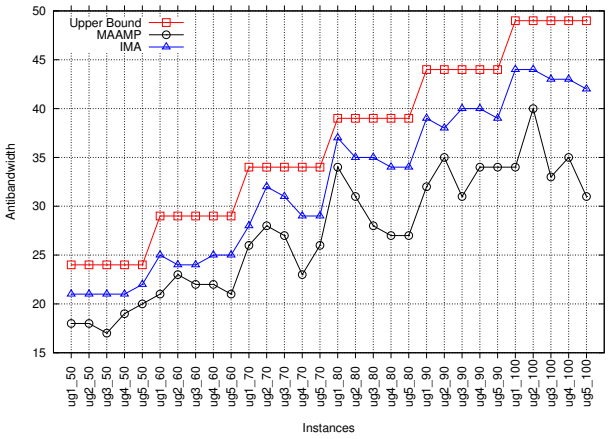

Fig. 2. Performance comparison between MAAMP and IMA with respect to the theoretical upper bounds.

tunning methodology based on Combinatorial Interaction Testing [7], to yield the best solution quality in a reasonable computational time.

The practical usefulness of this fine-tunned IMA algorithm was assessed with respect to an existing state-of-the-art Memetic Algorithm, called MAAMP [14] over a set of 30 well-known benchmark graphs taken from the literature. The results show that our IMA algorithm was able to consistently produce labelings with higher antibandwidth values than those furnished by MAAMP. Furthermore, IMA achieves those results by employing only a small fraction (10.01%) of the total time used by MAAMP.

This work opens up a range of possibilities for future research. Currently we are interested on developing a Multimeme Algorithm [26] based on the Memetic Algorithm presented here in order to efficiently solve the antibandwidth problem for bigger graphs with different topologies.

References

1. Leung, J., Vornberger, O., Witthoff, J.: On some variants of the bandwidth minimization problem. SIAM Journal on Computing **13**(3) (1984) 650–667
2. Yixun, L., Jinjiang, Y.: The dual bandwidth problem for graphs. Journal of Zhengzhou University **35**(1) (March 2003) 1–5
3. Hale, W.K.: Frequency assignment: Theory and applications. Proceedings of the IEEE **68**(12) (December 1980) 1497–1514
4. Roberts, F.S.: New directions in graph theory. Annals of Discrete Mathematics **55** (1993) 13–44
5. Cappanera, P.: A survey on obnoxious facility location problems. Technical report, Uni. di Pisa (1999)
6. Burkard, R.E., Donnani, H., Lin, Y., Rote, G.: The obnoxious center problem on a tree. SIAM Journal on Computing **14**(4) (2001) 498–509

7. Cohen, D.M., Dalal, S.R., Parelius, J., Patton, G.C.: The combinatorial design approach to automatic test generation. IEEE Software **13**(5) (1996) 83–88
8. Miller, Z., Pritikin, D.: On the separation number of a graph. Networks **19** (1989) 651–666
9. Yao, W., Ju, Z., Xiaoxu, L.: Dual bandwidth of some special trees. Journal of Zhengzhou University Natural Science Edition **35** (2003) 16–19
10. Calamoneri, T., Missini, A., Török, L., Vrt'o, I.: Antibandwidth of complete k-ary trees. Electronic Notes in Discrete Mathematics **24** (2006) 259–266
11. Török, L.: Antibandwidth of three-dimensional meshes. Electronic Notes in Discrete Mathematics **28** (March 2007) 161–167
12. Raspaud, A., Schröder, H., Sykora, O., Török, L., Vrt'o, I.: Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discrete Mathematics **309**(11) (2009) 3541–3552
13. Wang, X., Wu, X., Dimitrescu, S.: On explicit formulas for bandwidth and antibandwidth of hypercubes. Discrete Applied Mathematics **157**(8) (2009) 1947–1952
14. Bansal, R., Srivastava, K.: Memetic algorithm for the antibandwidth maximization problem. Journal of Heuristics **17**(1) (2011) 39–60
15. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. 1st edn. Springer (2007)
16. Davis, L.: Applying adaptive algorithms to epistatic domains. In: Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1985) 162–164
17. Goldberg, D.E., Lingle, R.: Alleles, loci, and the travelling salesman problem. In: Proceedings of the 1st. International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates (1985) 154–159
18. Oliver, I.M., Smith, D.J., Holland, J.R.C.: A study of permutation crossover operators on the travelling salesman problem. In: Proceedings of the 2nd. International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates (1987) 224–230
19. Freisleben, B., Merz, P.: A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: Proceedings of IEEE International Conference on Evolutionary Computation, IEEE Press (1996) 616–621
20. Adenso-Diaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental design and local search. Operations Research **54**(1) (2006) 99–114
21. de Landgraaf, W.A., Eiben, A.E., Nannen, V.: Parameter calibration using meta-algorithms. In: In proceedings of the IEEE Congress on Evolutionary Computation, IEEE Press (2007) 71–78
22. Gunawan, A., Lau, H.C., Lindawati: Fine-tuning algorithm parameters using the design of experiments. Lecture Notes in Computer Science **6683** (2011) In press
23. Gonzalez-Hernandez, L., Torres-Jimenez, J.: MiTS: A new approach of tabu search for constructing mixed covering arrays. Lecture Notes in Artificial Intelligence **6438** (2010) 382–392
24. Colbourn, C.J.: Combinatorial aspects of covering arrays. Le Matematiche **58** (2004) 121–167
25. Rodriguez-Tello, E., Torres-Jimenez, J.: Memetic algorithms for constructing binary covering arrays of strength three. Lecture Notes in Computer Science **5975** (2010) 86–97
26. Krasnogor, N.: Towards robust memetic algorithms. In: Recent Advances in Memetic Algorithms. Volume 166 of Studies in Fuzziness and Soft Computing. Springer (2004) 185–207