Comparative Study of Different Memetic Algorithm Configurations for the Cyclic Bandwidth Sum Problem

Eduardo Rodriguez-Tello[1][0000−0002−0333−0633], Valentina Narvaez-Teran[1][0000−0003−2071−9568], and Fréderic Lardeux[2][0000−0002−0907−7886]

[1] CINVESTAV – Tamaulipas.
Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria Tamps., Mexico
{ertello, mnarvaez}@tamps.cinvestav.mx
[2] LERIA, Université d'Angers. 2 Boulevard Lavoisier, 49045 Angers, France
frederic.lardeux@univ-angers.fr

Abstract. The Cyclic Bandwidth Sum Problem (CBSP) is an NP-Hard Graph Embedding Problem which aims to embed a simple, finite graph (the guest) into a cycle graph of the same order (the host) while minimizing the sum of cyclic distances in the host between guest's adjacent nodes. This paper presents preliminary results of our research on the design of a Memetic Algorithm (MA) able to solve the CBSP. A total of 24 MA versions, induced by all possible combinations of four selection schemes, two operators for recombination and three for mutation, were tested over a set of 25 representative graphs. Results compared with respect to the state-of-the-art top algorithm showed that all the tested MA versions were able to consistently improve its results and give us some insights on the suitability of the tested operators.

Keywords: Cyclic Bandwidth Sum Problem · Memetic Algorithms · Graph Embedding Problems

1 Introduction

Graph Embedding Problems (GEP) are combinatorial problems which aim to find the most suitable way to embed a guest graph G into a host graph H [3, 5]. An embedding is a labeling of the vertices of G by using the vertices of H. The Cyclic Bandwidth Sum Problem [2] can be formally defined as follows. Let $G = (V, E)$ be a finite undirected (guest) graph of order n and C_n a cycle (host) graph with vertex set $|V_H| = n$ and edge set E_H. Given an injection $\varphi : V \rightarrow V_H$, representing an embedding of G into C_n, the cyclic bandwidth sum (the cost) for G with respect to φ is defined as:

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

where $|x|_n = \min\{\, |x|,\ n - |x| \,\}$ (with $1 \leq |x| \leq n - 1$) is called the *cyclic distance*, and the label associated to vertex u is denoted $\varphi(u)$.

Then, the CBSP consists of finding the optimal embedding φ^*, such that $\mathrm{Cbs}(G, \varphi^*)$ is minimum, i.e., $\varphi^* = \arg\min_{\varphi \in \Phi}\{\mathrm{Cbs}(G, \varphi)\}$ with Φ denoting the set of all possible embeddings.

The CBSP is an NP-Hard problem originally studied by Yuang [16]. Most of the work reported in the literature has focused on theoretical research about calculating (or at least approximating) the optimal solution for some well-known graph topologies. Some of the topologies addressed by the reported exact formulas [2] are paths, cycles, wheels, k-th powers of cycles and complete bipartite graphs. For the Cartesian products of two graphs (when those graphs are paths, cycles or complete graphs) upper bounds have been reported in [9]. The relation of the CBSP with the Bandwidth Sum Problem[3] (BSP) was also studied [2]. Given the relevant applications of this problem on VLSI designs [1, 15], code design [7], simulation of network topologies for parallel computer systems [12], scheduling in broadcasting based networks [11], signal processing over networks [6] and compressed sensing in sensor networks [10], it has recently caught attention in the combinatorial optimization and operation research areas.

Theoretical formulations are useful to estimate optimal values, but they say little about how to algorithmically construct optimal embeddings, or at least near optimal solutions. This resulted in the development of two approximated algorithms devised to solve the CBSP: General Variable Neighborhood Search (GVNS) [14] and a greedy heuristic denominated as MACH [6].

GVNS algorithm applies Reduced Variable Neighborhood Search (RVNS) to improve its initial solution, which consist of a lexicographical embedding. The properly said GVNS phase includes six perturbation operators and two neighborhoods. When dealing with path, cycle, star and wheel topologies of order $n \leq 200$ GVNS was able to achieve optimal results as well as solutions under the theoretical upper bounds for Cartesian products of order $n \leq 64$ and graphs of the Harwell-Boeing collection of order $n \leq 199$.

MACH is a two phase greedy heuristic algorithm. In the first phase the guest graph G is partitioned into disjoint paths by a depth first search mechanism guided by the Jaccard index [8] as a similarity criterion between vertices. Since Jaccard index measures the similarity between vertices neighborhoods, vertices with common neighbors are likely to be included near each other in the same path. In the second phase a solution is incrementally built up by merging the paths. The longer path is added to the solution, then a greedy strategy is implemented to determine where in the partial solution the remaining paths should be inserted. It was experimentally shown that MACH consistently improves the solution quality achieved by GVNS, as well as the running time. Therefore, MACH is currently considered as the best-known algorithm to solve the CBSP.

Our approach consists in studying a combination of genetic and local search inspired operators implemented into a Memetic Algorithm to solve the general case of the CBSP. We worked with four selection schemes, two recombination

[3] BSP is the problem of embedding a graph into a path while minimizing the sum of linear distances between embedded vertices.

mechanisms, three mutation schemes and one survival strategy. The 24 possible combinations of operators (MA versions) were duly tested.

Our experiments over a set of 25 topologically diverse representative instances allowed us to obtain significantly improved results with respect to the state-of-the-art top algorithm. We also obtained some insights about the effectiveness of some of the tested operators for helping solving the CBSP.

The rest of this work is organized as follows. MA main routine and operator implementations are described in Sect. 2. Our experimental methodology and the results of the comparisons among the 24 implemented MA versions with respect to the literature results are shown and discussed in Sect. 3. Finally, the conclusions of this work and further research directions are presented in Sect. 4.

2 Memetic Algorithms for the CBSP

Algorithm 1 describes the main framework common to all our MA versions. Population P contains μ individuals. At each generation we chose from P couples of individuals for recombination by crossover. Then, the resulting individuals are mutated and extra perturbations of their chromosomes are performed by inversion. Local search is applied only to the best individual P_{best} in the surviving population P, in order to accelerate the computational time expended in each generation. Furthermore, as it is described in Sect. 2.4, the mutation operators also incorporate certain local search operations.

Although o'' is the individual added to the offspring population O, we also compare the fitness corresponding to previous states of its chromosome (o and o') with the best historically found solution g, in order to avoid losing any possible improvement, even if o and o' are not actually in O. The historically best found solution record g is kept independently of the populations P and O.

2.1 Solution Encoding and Initialization

The potential solutions were turned into chromosomes by the permutation encoding. An individual is represented as $P_i = (\varphi_i, \rho_i, f_i)$ where φ_i and ρ_i are two representations of the same embedding: $\varphi_i(u)$ stands for the label associated to vertex u (i.e., the vertex in the host graph associated to vertex u). $\rho_i(u')$ denotes the vertex in G having the label u' (i.e., the vertex hosted in vertex u'); and $f_i = f(\varphi_i, G)$ is the fitness of the individual assessed by the fitness function which corresponds to (1). Whenever a change occurs in φ_i it is reflected in ρ_i and vice-versa. All individuals in population P are initialized by the assignment of random permutations to their chromosomes. With exception of *insertion* mutation, all of our operators work primarily over φ_i.

2.2 Selection

We will denote S as a multiset containing the individuals for mating. Since we use the Cbs values as fitness values and CBSP is a minimization problem, the individuals with lower Cbs values are actually the fittest ones. Therefore, in the

Algorithm 1: Memetic Algorithm

```
1: P ← initializePopulation(P, μ)
2: O ← ∅
3: t ← 1
4: g ← P_best
5: repeat
6:    for i ← 1 to μ do
7:       P_a, P_b ← selection(P)
8:       o ← crossover(P_a, P_b, prob_c)
9:       o' ← mutation(o, prob_m)
10:      o'' ← inversion(o', prob_i)
11:      O ← O ∪ o''
12:      g ← fitter individual among current g, o, o' and o''
13:   end for
14:   P ← survival(P, O)
15:   O ← ∅
16:   P_best ← localsearch(P_best, tries)
17:   g ← fitter individual among current g and P_best
18: until stop criterion is met
19: return g
```

case of *stochastic* and *roulette* selections we performed a min-max normalization of the fitness values. Then, for each individual its expected value was calculated based on its normalized fitness.

Stochastic selection is performed by adding to S as many copies of each individual as the integer part of its expected value indicates. Then, the floating point parts are used to probabilistically determine whether or not to add an additional copy.

In *roulette* selection the expected values serve as an indicator of the size of the section corresponding to each individual in the roulette. We pick 2μ individuals by spinning the roulette 2μ times. The higher the expected values, the bigger the section and the higher chances for the individual to be chosen.

Random selection is rather simple, it just picks 2μ individuals from P, with replacement. *Binary tournament* performs 2μ tournament rounds. At each round the individual with the lower Cbs value is chosen. So, when implementing *random* or *binary tournament* selections there is neither need for normalization nor for expected values.

2.3 Crossover

Two permutation specialized crossover operators were implemented: *cyclic* [13] and *order-based* crossover [4]. An offspring is created as follows. First, a couple of individuals from S is picked with replacement. Each couple can produce only one offspring. It is probabilistically decided if this individual is created by recombination, with probability $prob_c$, or if it is a copy of the fitter individual in the selected couple.

Cyclic crossover operates by computing the *cycles* between both parent chromosomes. The individual inherits, alternately, one cycle from one of the parents and one from the other. By doing this, the operator produces a new permutation in which the absolute positions of each of its genes is preserved with respect to one of the parents, and therefore implicit mutations are avoided.

Order-based crossover picks a random segment of genes from one parent individual and inherits it directly to the offspring. Then, the rest of genes of the offspring are assigned in the same order as they appeared in the other parent. This operator balances the preserving of absolute positions of the permutation elements and their relative order. It introduces implicit mutations, but within a limited scope.

2.4 Mutation

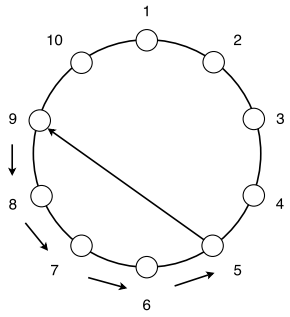
Keeping the population diverse is necessary to avoid premature convergence. Diversification is provided by mutation, introducing new genetic material into the population. Mutation works by probabilistically altering some of the genes of an individual. We tested three existing mutation schemes for permutations: *insertion*, *reduced 3-swap* and *cumulative swap*.

Insertion mutation operates over ρ_i (see Sect. 2.1). By manipulating ρ_i, *insertion* models the process of reallocating the guest vertex embedded at the host vertex u' to the vertex v', while displacing the embedded vertices between u' and v'. Both host vertices u' and v' are randomly chosen. Given the cyclic nature of the embeddings for the CBSP, there are actually two sections of vertices that can be considered to be the *section in between* u' and v': one section implies clockwise displacements and the other counterclockwise displacements. The insertion mutation will affect only the smaller section, i.e., the one with fewest vertices, which corresponds to the minimum length path between u' and v' in C_n. Figure 1 illustrates this by representing ρ_i as a cyclic permutation in order to reflect the cyclic nature of the embedding it encodes. As it can be inferred, any change in ρ_i must be properly reflected in φ_i by updating the labels, i.e., host vertices of the guest vertices embedded in the affected section.

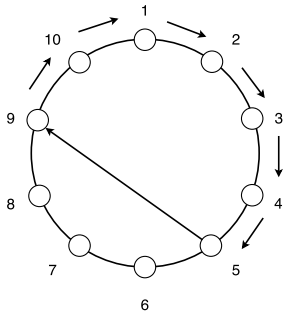
Reduced 3-swap mutation picks three random vertices. The labels of those nodes are exchanged in every possible way, giving as a result five new solutions. The individual is then replaced by the best of those solutions, even if its fitness is worse than the current one. This can be seen as a subneighborhood from the 3-swap neighborhood, i.e., all solutions at Hamming distance equal to three from the current solution.

Cumulative swap performs $n/2$ iterations (steps). At each iteration, with probability $prob_c$ a pair of random vertices is picked. It is evaluated if the fitness of the individual would be improved by exchanging the labels of those vertices. If so, the labels are actually exchanged. Cumulative swap can be seen as a random up-hill walk of limited length.

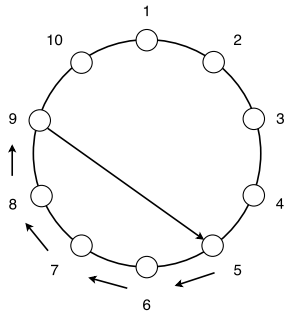
One of the differences in the application of one or other mutation scheme is the role of the mutation probability $prob_m$. In the case of *reduced 3-swap* and *insertion*, the mutation probability acts at individual level, i.e., it is decided only



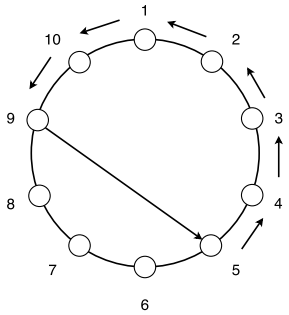
(a) Counterclockwise displacements for $u' = 5$ and $v' = 9$.



(b) Clockwise displacements for $u' = 5$ and $v' = 9$.



(c) Clockwise displacements for $u' = 9$ and $v' = 5$.



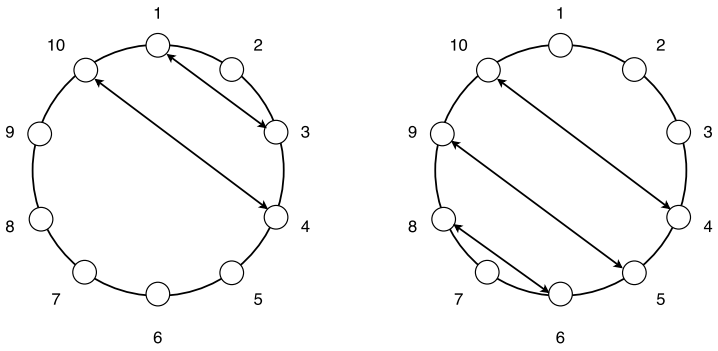
(d) Counterclockwise displacements for $u' = 9$ and $v' = 5$.

Fig. 1. *Insertion* mutation. Numbers represent the vertices of permutation ρ_i. Example in Fig. 1(a) corresponds to counterclockwise insertion when $u' < v'$, performing 5 steps. Also for $u' < v'$, clockwise insertion will perform 7 steps, as shown in Fig. 1(b), therefore counterclockwise insertion is preferred. For $u' > v'$, clockwise insertion will perform 5 steps, while 7 steps will be required by counterclockwise insertion.

once per generation if an individual will be mutated or not. Meanwhile, in *cumulative swap* little probabilistic mutations occur up to $n/2$ times per individual. From this follows that, when using *insertion* or *reduced 3-swap* mutations some individuals will remain unchanged, approximately $1 - prob_m \cdot \mu$. The mutated individuals will present variable size mutations in the case of *insertion*, and uniform size mutations (exactly 3) in the case of *reduced 3-swap*. In *cumulative swap* it is likely that all individuals will mutate, but the amount of genes affected will vary within the population.

2.5 Inversion

The *inversion* phase is independent of the mutation one. In a similar way to the *reduced 3-swap* and *insertion* mutations, it is probabilistically applied at individual level, so some individuals could remain unchanged. Given the nature



(a) Clockwise inversion.

(b) Counterclockwise inversion.

Fig. 2. Inversion over φ_i, numbers represent the permutation labels. In Fig. 2(a) a clockwise inversion between vertices $u = 10$ and $v = 4$ would perform two exchanges of labels. Figure 2(b) shows the respective counterclockwise inversion which performs three exchanges of labels, therefore clockwise inversion is preferred.

of this operator, the number of changed genes in the affected individuals will be variable. *Inversion* operator consists in selecting two random vertices, and reversing the order of appearance of the labels in the section between them (inclusively). This is achieved by consecutive exchanges in the φ_i representation. In Fig. 2, φ_i is represented as a cyclic permutation to illustrate this process. Similarly to *insertion*, the cyclic feature of CBSP embeddings is considered, and *inversion* can operate clockwise or counterclockwise, preferring always the option implying the minimal number of exchanges.

2.6 Survival Strategy

The survival strategy applied was $(\mu + \lambda)$. All individuals in populations P and O are merged and sorted in nondecreasing order according with their fitness values. Then, the first μ individuals are chosen to become the parent population P for the next generation.

2.7 Local Search

Local search is applied only to the best individual in the survivor population. The neighborhood employed was the one induced by the 2-swap operator, i.e., all solutions resulting from swapping the labels of two vertices in φ_i. It is visited in a random order, using the first-improvement move strategy. The local search phase ends when a local optimum is reached or after a maximal number of iterations was performed (*tries*).

Table 1. Input parameter values for the MA algorithms.

Parameter		Value	Parameter		Value
Population size	μ	20	Inversion rate	$prob_i$	0.240
Crossover rate	$prob_c$	0.788	Local search iterations	$tries$	10
Mutation rate	$prob_m$	0.543	Evaluation function calls	T	4.0E+08

3 Experimental Results

We experimented with the full set of 24 MA versions corresponding to all the possible combinations of operators, with a maximal number (T) of calls to the fitness function as stop criterion. A set of 25 topologically diverse and representative instances (see Table 3) belonging to three different types was used: Cartesian products, paths and cycles, and Harwell-Boeing graphs. All the MA versions were tested using a fixed set of parameter values (Table 1) obtained from the literature and from our a priori experiments using the *irace* R package for automatized algorithm tunning. Details on this matter are no included here due to the space limitations, but they are available online.[4]

For comparing the algorithms in terms of solution quality the overall relative root mean square error (O-RMSE) was computed for $R = 31$ runs, with respect to the best-known solutions for the $|\mathcal{T}| = 25$ tested instances, see (2). Those solutions were provided either by MACH or by any of our 24 memetic algorithms. The O-RMSE among all instances $t \in \mathcal{T}$ was calculated as:

$$\text{O-RMSE} = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} 100\%\sqrt{\left(\sum_{r=1}^{R}\left(\frac{\text{Cbs}_r(t)-\text{Cbs}^*(t)}{\text{Cbs}^*(t)}\right)^2\right)/R}\ , \qquad (2)$$

where $\text{Cbs}_r(t)$ is the best solution quality achieved by the algorithm at execution r, and $\text{Cbs}^*(t)$ is the best-known quality solution for instance $t \in \mathcal{T}$. An O-RMSE equal to 0% means the algorithm achieved the best known solution quality in all the R executions, and therefore it is the preferred value.

We also performed statistical significance analysis by the following methodology. The normality of data distributions was evaluated by the *Shapiro-Wilk* test. *Bartlett's* test was implemented to determine whether the variances of the normally distributed data were homogeneous or not. *ANOVA* test was applied in the case variance homogeneity was present and *Welch's t* parametric tests on the contrary. Meanwhile, *Kruskal-Wallis* test was implemented for non-normal data. In all cases the significance level considered was 0.05.

In order to identify the combination of operators corresponding to the different memetic algorithms we assigned keys to the tested operators, then those keys were used to construct a unique MA configuration identifier. The operator keys are *a)* for selection: *stochastic* (S1), *roulette* (S2), *random* (S3) and *binary tournament* (S4); *b)* for crossover: *cyclic* (C1) and *order-based* (C2); and *c)* for

[4] `http://www.tamps.cinvestav.mx/~ertello/cbsp-ma.php`

Table 2. Results for the 24 MA tested versions.

#	Algorithm	Op. configuration	O-RMSE (%)	Avg. ex. time (s)	G_{best} time (s)
1	MA-10	S2_C2_M1	5.297	87.434	19.886
2	MA-22	S4_C2_M1	5.637	87.133	20.623
3	MA-16	S3_C2_M1	5.854	86.427	19.954
4	MA-04	S1_C2_M1	5.945	87.569	19.728
5	MA-19	S4_C1_M1	6.030	86.911	17.741
6	MA-13	S3_C1_M1	6.609	86.230	17.942
7	MA-07	S2_C1_M1	6.693	87.238	18.808
8	MA-01	S1_C1_M1	6.715	87.339	18.587
9	MA-05	S1_C2_M2	10.022	85.434	21.414
10	MA-17	S3_C2_M2	10.065	84.391	23.983
11	MA-23	S4_C2_M2	10.237	85.064	22.840
12	MA-11	S2_C2_M2	10.583	85.353	22.958
13	MA-14	S3_C1_M2	10.626	84.556	22.129
14	MA-02	S1_C1_M2	11.092	85.573	22.103
15	MA-08	S2_C1_M2	11.383	85.523	21.973
16	MA-20	S4_C1_M2	11.389	85.223	21.235
17	MA-06	S1_C2_M3	12.749	97.562	24.883
18	MA-12	S2_C2_M3	12.872	97.463	24.396
19	MA-18	S3_C2_M3	13.182	96.630	23.779
20	MA-24	S4_C2_M3	13.449	97.210	25.685
21	MA-09	S2_C1_M3	14.116	97.191	23.832
22	MA-03	S1_C1_M3	14.285	97.305	23.219
23	MA-21	S4_C1_M3	15.067	96.952	22.853
24	MA-15	S3_C1_M3	15.077	96.377	24.367
25	MACH	N/A	21.050	2.09	N/A

mutation: *insertion* (M1), *reduced 3-swap* (M2) and *cumulative swap* (M3). Since all versions consider only $(\mu + \lambda)$ as survival strategy there is no need to assign a key for it.

Table 2 presents our algorithms ranked according to their performance in terms of solution quality. MACH, which ranked last after all the MA, is also included as reference. Table 2 includes the rank of the algorithm (#), the configuration of genetic operators, the associated O-RMSE value, the average total running time (in seconds) and the average time in which the reported best found solution was reached by the algorithm. Since MACH is a constructive approach, only its average total time is reported.

The results in Table 2 suggest that the recombination and mutation schemes are more decisive than the selection, since the former operators induce the most remarkable grouping, indicated by the dashed lines. Despite algorithms including order-based crossover being better performing than their counterparts implementing cyclic crossover, it is the mutation operator the one having the higher influence over the final solution quality reached by the MA. Focusing on O-RMSE values, we found that the wider performance gap (of almost 5% O-RMSE) is observed between the algorithms implementing *insertion* mutation (M1) and the rest, while the gap between *cyclic* crossover (C1) or *order-based* crossover (C2) rarely surpasses 1%. From Table 2 it can be inferred that the top 3 MA configurations are quite similar in solution quality, total running time and time to find their best solution. The statistical significance analysis showed that, for the instance set being tested, our top 3 MA configurations are statistically indis-

tinguishable from each other in terms of solution quality. This is not surprising since they differ only in the selection scheme. Moreover, the three of them are able to provide better solutions than MACH. Even the worst performing of our MA versions (MA-15) can provide better solutions than MACH. MA-15 has a O-RMSE value of 15.077%, meanwhile the O-RMSE of MACH surpasses 20%.

Although all the Memetic Algorithms take longer time than MACH, it is worth noting that MACH solution quality cannot be improved by employing a longer running time. It is also observable that all of our algorithms stopped finding improving solutions at an early stage of their total running time. Since there are some instances for which the optimal solutions or upper bounds were not always reached, this may be an indicator of premature convergence. While mutation and inversion are diversification mechanisms their effect may be diluted by the survival strategy. Once a locally optimal individual is reached, it will remain in the population in next generations and its genes are likely to keep proliferating in the population, until a fitter individual appears. Meanwhile, the less fit individuals will disappear from the population and diversity may be lost in preference of individuals becoming (probably) a locally optimal solution.

Table 3 presents the results of MA-10, the one with the best performance, compared with the state of the art. Only MACH is considered for the comparison, since it has been experimentally shown better than GVNS [6]. For each of the 25 instances in the set we present its number of vertices ($|V|$), number of edges ($|E|$), density ($d = 2|E|/|V|(|V|-1)$) and value of the optimum or upper bound (UB/Opt*). Those values were assessed according to the graph topology: upper bound formula for the Cartesian products [9]; optimal value formula (marked by the symbol *) for path, cycle, wheel and k-th power of cycle topologies [2, 9], and the general graph upper bound formula [9] for the Harwell-Boeing graphs.

Our best MA is compared to MACH [6], including the minimal of the solution cost values (*Best*) found among 31 executions, average and standard deviation of the those values (*Std*), and average time to reach the reported solutions. The last column (MA-10/MACH) corresponds to the result of the statistical significance test performed. Instances where MA-10 results present improvements with statistical significance with respect to those achieved by MACH are indicated by the $+$ symbol, meaning a *victory* for MA-10. The contrary case, a *defeat* for MA-10, is marked with the $-$ symbol. Results with no statistical significant difference are counted as ties and marked with the $\star$ symbol. The best known solution for each instance is highlighted in bold.

For most of the tested instances MA-10 is able to consistently produce significantly better solutions with respect to those furnished by MACH. Our only defeats correspond to graphs for which MACH is specially suitable to solve: highly regular topologies with low densities, such as cycles and paths. However, MA-10 shows dominance for regular topologies with growing densities (see Cartesian products) and more general graphs, such as Harwell-Boeing graphs. It is also noticeable that our algorithm reached solutions with CBS values under the theoretical upper bounds (or equal to the optimal know values) for all instances.

Table 3. Performance comparison of our best performing MA (MA-10), with respect to the state-of-the-art method.

					Mach				MA-10 (S2_C2_M1)				MA-10
Graph	$\|V\|$	$\|E\|$	d	UB/Opt^*	*Best*	*Avg*	*Std*	T	*Best*	*Avg*	*Std*	T	/Mach
p9p9	81	144	0.04	720	944	1254.77	183.07	0.00	**516**	585.68	96.65	3.51	+
c9c9	81	162	0.05	873	991	1283.65	131.95	0.01	**873**	961.52	85.73	6.30	+
p9c9	81	153	0.05	7434	794	794.00	0.00	0.00	**745**	805.81	73.38	5.62	⋆
p9k9	81	396	0.12	7362	**1728**	1728.00	0.00	0.01	**1728**	1728.00	0.00	1.13	⋆
c9k9	81	405	0.13	7434	**1809**	1809.00	0.00	0.01	**1809**	1809.00	0.00	0.68	⋆
k9k9	81	648	0.20	8370	9454	9533.32	43.63	0.02	**8280**	8605.81	270.05	21.87	+
path100	100	99	0.02	99*	**99**	99.00	0.00	0.00	**99**	99.00	0.00	7.48	⋆
cycle100	100	100	0.02	100*	**100**	100.00	0.00	0.00	**100**	144.65	56.29	5.13	−
wheel100	100	198	0.04	2600*	**2600**	2600.00	0.00	0.01	**2600**	2633.42	45.94	11.71	−
cPow100-10	100	1000	0.20	5500*	5598	5703.74	68.71	0.04	**5500**	5500.00	0.00	11.89	+
cPow100-2	100	200	0.04	300*	**300**	302.52	2.42	0.00	**300**	385.16	155.97	5.16	⋆
can_24	24	68	0.25	425	220	255.03	16.01	0.01	**182**	182.00	0.00	0.18	+
ibm32	32	90	0.18	743	493	540.35	22.94	0.01	**405**	411.84	8.18	1.84	+
bcspwr01	39	46	0.06	460	102	115.58	8.53	0.01	**98**	102.58	5.82	4.80	+
bcsstk01	48	176	0.16	2156	1157	1339.74	111.74	0.02	**936**	954.45	13.43	21.32	+
bcspwr02	49	59	0.05	737	158	176.23	20.03	0.02	**148**	151.94	5.93	10.53	+
curtis54	54	124	0.09	1705	448	633.61	89.46	0.03	**411**	422.90	20.66	8.54	+
will57	57	127	0.08	1841	408	436.55	45.42	0.04	**335**	345.29	21.55	0.60	+
impcol_b	59	281	0.16	4215	2462	2838.13	242.00	0.07	**1822**	1829.74	9.90	0.16	+
ash85	85	219	0.06	4708	1232	1422.16	142.17	0.14	**919**	1036.58	89.64	22.89	+
nos4	100	247	0.05	6237	1181	1397.48	222.87	0.07	**1031**	1031.00	0.00	6.03	+
bcspwr03	118	179	0.03	5325	766	926.90	76.74	0.25	**664**	713.19	53.72	14.67	+
can_292	292	1124	0.03	82333	23288	25703.48	1678.87	7.13	**15763**	18982.10	2148.92	75.81	+
bcsstk06	420	3720	0.04	391532	65017	84469.87	8027.79	30.83	**55140**	67875.65	10377.12	177.82	+
impcol_d	425	1267	0.01	134935	25677	35355.19	4596.68	13.48	**12232**	15932.90	3170.52	71.47	+

Note: The overall winner MA-10 scored 18 victories (+), 2 defeats (−), and 5 ties (⋆).

4 Conclusions and Future Work

A set of 24 different MA configurations for solving the CBSP was evaluated. The experiments presented revealed that the top three MA configurations, which are statistically indistinguishable from each other, can provide significantly better results than Mach [6] for 18 out of 25 tested instances. Furthermore, the best MA version (MA-10) achieved optimal results for the 5 instances with known exact solution values. For the remaining 20 instances with unknown exact optimal values, MA-10 was able to establish 18 new upper bounds and to equal 2 other.

Confirming the presence of premature convergence in our MA, as well as identify its causes, are certainly interesting future research topics. Exploring other alternatives for the survival strategy, as well as using Mach as an initialization operator, could be promising directions to improve the performance our MA. It is also interesting to consider the implementation of automatic schemes allowing the algorithm to self-adapt its own operators, instead of defining them from the beginning of the search.

Acknowledgments. The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies at CINVESTAV-Tamaulipas.

References

1. Bhatt, S.N., Leighton, F.T.: A framework for solving VLSI graph layout problems. Journal of Computer and System Sciences **28**(2), 300–343 (1984). https://doi.org/10.1016/0022-0000(84)90071-0
2. Chen, Y., Yan, J.: A study on cyclic bandwidth sum. Journal of Combinatorial Optimization **14**(2), 295–308 (2007). https://doi.org/10.1007/s10878-007-9051-y
3. Chung, F.R.K.: Labelings of graphs. In: Beineke, L.W., Wilson, R.J. (eds.) Selected Topics in Graph Theory, vol. 3, chap. 7, pp. 151–168. Academic Press (1988)
4. Davis, L.: Applying adaptive algorithms to epistatic domains. In: Proceedings of the 9th IJCAI. vol. 1, pp. 162–164. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1985)
5. Diaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Computing Surveys **34**(3), 313–356 (2002). https://doi.org/10.1145/568522.568523
6. Hamon, R., Borgnat, P., Flandrin, P., Robardet, C.: Relabelling vertices according to the network structure by minimizing the cyclic bandwidth sum. Journal of Complex Networks **4**(4), 534–560 (2016). https://doi.org/10.1093/comnet/cnw006
7. Harper, L.: Optimal assignment of numbers to vertices. Journal of SIAM **12**(1), 131–135 (1964). https://doi.org/10.1137/0112012
8. Jaccard, P.: The distribution of the flora in the alpine zone. New Phytologist **11**(2), 37–50 (1912). https://doi.org/10.1111/j.1469-8137.1912.tb05611.x
9. Jianxiu, H.: Cyclic bandwidth sum of graphs. Applied Mathematics - A Journal of Chinese Universities **16**(2), 115–121 (2001). https://doi.org/10.1007/s11766-001-0016-0
10. Li, Y., Liang, Y.: Compressed sensing in multi-hop large-scale wireless sensor networks based on routing topology tomography. IEEE Access (2018). https://doi.org/10.1109/ACCESS.2018.2834550
11. Liberatore, V.: Multicast scheduling for list requests. In: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies. vol. 2, pp. 1129–1137. IEEE (2002). https://doi.org/10.1109/INFCOM.2002.1019361
12. Monien, B., Sudborough, I.H.: Embedding one interconnection network in another, vol. 7, pp. 257–282 (1990). https://doi.org/10.1007/978-3-7091-9076-0_13
13. Oliver, I., Smith, D., Holland, J.: A study of permutation crossover operators on the traveling salesman problem. In: Proceedings of the 2nd International Conference on Genetic Algorithms and Their Application. pp. 224–230. L. Erlbaum Associates Inc., Hillsdale, NJ, USA (1987)
14. Satsangi, D., Srivastava, K., Gursaran: General variable neighbourhood search for cyclic bandwidth sum minimization problem. In: Proceedings of the Students Conference on Engineering and Systems. pp. 1–6. IEEE Press (March 2012). https://doi.org/10.1109/SCES.2012.6199079
15. Ullman, J.D.: Computational Aspects of VLSI. Computer Science Press (1984)
16. Yuan, J.: Cyclic arrangement of graphs. Graph Theory Notes of New York, New York Academy of Sciences pp. 6–10 (1995)