A Comparison of Memetic Recombination Operators 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. In this paper the Minimum Linear Arrangement (MinLA) problem is studied within the framework of memetic algorithms (MA). A new dedicated recombination operator called Trajectory Crossover (TX) is introduced and its performance is compared with four previous crossover operators. It is shown that the TX crossover induces a better population diversity. The MA using TX is evaluated on a set of well-known benchmark instances and is compared with several state-of-art MinLA algorithms.

Key words: Recombination Operators, Memetic Algorithms, Linear Arrangement.

1 Introduction

The *Minimum Linear Arrangement* problem (MinLA) was first stated by Harper in [11]. His aim was to design error-correcting codes with minimal average absolute errors on certain classes of graphs. MinLA arises also in other application areas like graph drawing, VLSI layout, software diagram layout and job scheduling [3].

MinLA can be stated formally as follows. Let $G(V, E)$ be a finite undirected graph, where V ($|V| = n$) defines the set of vertices and $E \subseteq V \times V = \{\{i, j\}|i, j \in V\}$ is the set of edges. Given a one-to-one 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.

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

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 [3] for a detailed survey). However, MinLA is NP-hard for general graphs [7] and for bipartite graphs [4]. 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) [1], the multi-scale algorithm (MS) [13] and the algebraic multigrid scheme (AMG) [18]; and b) metaheuristics such as Simulated Annealing [16] and Genetic Algorithms [17].

In this paper, we investigate the potential of the paradigm of Memetic Algorithms (MAs), which are known to be very powerful for hard combinatorial optimization problems [5, 6, 12]. In particular, we are interested in the design of an effective mechanism to recombine solutions which constitutes one of the key elements of MAs. A recombination operator, called Trajectory Crossover (TX), is introduced. This operator is based on the path relinking technique proposed in [8] and incorporates problem specific knowledge. Its performance is compared with four other classical crossover operators.

Our MA incorporates a fast greedy heuristic used to create the initial population and a local search operator based on a fine tuned Tabu Search algorithm. The effectiveness of the MA with TX is demonstrated 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 rest of the paper is organized as follows. Section 2 presents the different components of the MA used for the comparisons. Then, the studied recombination operators are reviewed in Section 3. Section 4 is dedicated to computational experiments and comparisons with previous reported results. Last section summarizes the main contributions of this research work.

2 Memetic Algorithms for MinLA

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

2.1 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 $i\text{-}th$ value $l[i]$ denotes the label assigned to the vertex i. The fitness of φ is evaluated by using Equation 1.

2.2 The General Procedure

Our MA 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 ($maxFails$).

2.3 The Initialization Operator

The operator $initPopulation(|P|)$ initiates the population P with $|P|$ configurations. To create a configuration, we use the binary balanced decomposition tree heuristic (DT) reported in [1], slightly adapted in order to work in a randomized form. The algorithm is based on a divide-and-conquer approach, the 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. Due to the randomness of the DT algorithm, the initial population is well diversified.

2.4 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 [5, 6, 14].

2.5 The Recombination Operator

The main idea of the recombination operator ($recombineIndividuals(A, B)$) is to generate new 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 Section 3, several recombination operators are presented, including the new TX recombination operator for MinLA.

2.6 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 our implementation, we have decided to use Tabu Search (TS) [8].

TS starts with a configuration, then it proceeds iteratively to visit a series of locally best configurations following a neighborhood function. At each iteration, a best neighbor is chosen to replace the current configuration, even if the former does not improve the current one. In order to avoid the stops at suboptimal points and the occurrence of cycles, TS introduces the notion of tabu list. The basic idea is to record each visited configuration, or generally its attributes and to forbid to visit again this configuration during the next T iterations (T is called the tabu tenure).

In our LS operator the neighbor of a given arrangement φ is obtained by swapping the labels of any pair (i, j) of different vertices. When such a move is performed the couple of vertices (i, j) is classified tabu for the next T iterations. Therefore, the vertices i and j cannot be exchanged during this period. Nevertheless, a tabu move leading to a configuration better than the best configuration found so far is always accepted (aspiration criterion). The tabu tenure T for a move is fixed to $0.10*n$. To implement the tabu list, it is sufficient to use an array of size $|V|$. The algorithm stops either if it reaches the predefined maximum of L iterations or when it ceases to make progress. In the proposed implementation a lack of progress exists when S successive iterations do not produce a better solution.

The algorithm memorizes and returns the most recent arrangement φ^* among the best configurations found: after each iteration, the current configuration φ replaces φ^* if $LA(G, \varphi) \leq LA(G, \varphi^*)$. It permits to produce a solution which is as far away as possible from the initial solution in order to better preserve the population diversity.

3 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 that can be applied to permutation problems [2, 5, 9, 15, 19]. In this section, we focus on four of these operators, as well as the new Trajectory Crossover dedicated to the MinLA problem.

3.1 Order Crossover

The Order Crossover (OX) operator was first proposed by Davis in [2]. It is implemented by selecting two random crossover points. The offspring inherits the elements between the two 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.

3.2 Partially Matched Crossover

The Partially Matched Crossover (PMX) operator was introduced in [9]. It is designed to preserve absolute positions from both parents. It works by selecting two crossover points in the first parent and copying the elements between them to the offspring. This transfer also defines a set of mappings between the elements that have been copied and the elements in the corresponding positions in the second parent. Then, the rest of the elements are copied in the positions they occur in the second parent. If one position is occupied by an element already copied from the first parent, the element provided by the mappings is considered. This process is repeated until the conflict is solved.

3.3 Cycle Crossover

Cycle Crossover (CX) is an operator that was proposed in [15]. It preserves the information contained in both parents in the sense that all elements of the offspring are taken from one of the parents, in other words CX does not perform any implicit mutation. In CX the offspring inherits all the elements found at the same position in the two parents. Then, starting with a randomly chosen unassigned position in the offspring, an element from one of the parents is randomly selected. After that, additional assignments are made to ensure that no implicit mutation occurs. Then, the next unassigned position to the right is processed in the same way until all the elements have been considered. In case there are still unassigned positions and we are at the end of the genome then we proceed at its beginning.

3.4 Distance Preserving Crossover

The Distance Preserving Crossover (DPX) operator reported in [5] relies on the notion of distance between solutions. DPX aims at producing an offspring that has the same hamming distance to each of its parents, and this distance is equal to the distance between the parents themselves. DPX starts by copying all the elements found at the same position in both parents to the offspring. Then, the rest of the positions in the offspring are randomly assigned with the yet unassigned elements, taking care that no assignment that is found in one of the parents is inherited into the child.

3.5 Trajectory Crossover

The new Trajectory Crossover (TX) for MinLA is inspired from the path relinking algorithm presented by Glover and Laguna as an alternative to integrate intensification and diversification strategies in the context of Tabu Search [8].

TX generates new offspring while exploring trajectories that connect two parents (A and B), by starting from one parent, called *initial solution*, and generating a trajectory in the neighborhood space that leads toward the alternate parent, called *guiding solution*. This process is accomplished by selecting moves

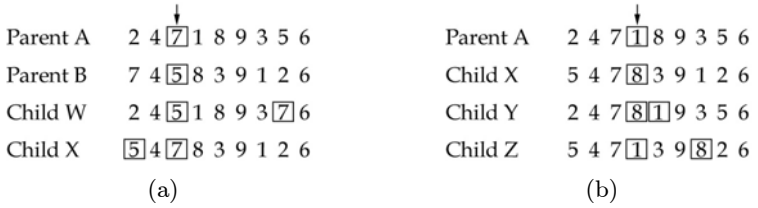


Fig. 1. Trajectory Crossover example

that introduce attributes contained in the guiding solution. Please note that each new solution in the trajectory corresponds to an individual.

In the TX operator the offspring inherits any element common to both the parents. Then, starting at a random position of the parents, their elements are examined from left to right in a cyclic fashion. If the elements at the position being looked at are the same, that position is skipped; otherwise, a swap is performed between two elements in parent A or in parent B, whichever gives the fitter solution, so that the elements at the analyzed position become alike. This process is repeated until all positions have been considered. All chromosomes obtained using this process are valid offspring of A and B; out of them the fittest offspring C is returned.

An example of the TX crossover is illustrated in Fig. 1. Suppose we start at a randomly chosen position, for this example position 3. In parent A, element 7 is located at the position 3 and in parent B element 5 is located at this position. There are two ways in which the two parents can move closer to one another; by swapping the elements 5 and 7 in parent A or in parent B (Fig. 1(a)). The fitness values of these two solutions are computed, suppose that child X has the lower cost. Then child W is eliminated and the next position in the two resulting solutions (A and X) is considered. In Fig. 1(b) the children obtained by swapping the elements 1 and 8 in parent A or in children X are presented. Their fitness is calculated and the child with higher cost is eliminated, the process is repeated until all positions have been considered.

4 Computational Experiments

In this section, we present a set of experiments accomplished to evaluate the performance of the different recombination operators presented in Section 3. Their characteristics were investigated by using them within the MA framework presented in Section 2. 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.

The test-suite used in the experiments presented in this paper is composed of the instances proposed by Petit[1] [16] and used later in [1, 13, 17]. It consists of six different families of graphs having a number of vertices between 62 and 9800.

4.1 Comparison of Recombination Operators

The purpose of the first experiment is to evaluate the performance of the different recombination operators presented in Section 3. The evaluation takes into account two aspects: the capacity to generate new potentially promising individuals and the ability to keep a diversified population. Both characteristics are very important in the whole search process because they represent the classical trade-off between exploration and exploitation. For the first criterion the average fitness is used, while the population diversity is calculated with the entropy measure proposed in [10] and shown in Equation 2, where n_{ij} represents the number of times the variable i is set to the value j in the population P. This function takes values in the interval $[0, 1]$. An entropy of 0 indicates that all the individuals in the population are identical.

$$entropy(P) = \frac{-\sum_{i=1}^{n}\sum_{j=1}^{n}(\frac{n_{ij}}{|P|})\log(\frac{n_{ij}}{|P|})}{n \log n} \quad (2)$$

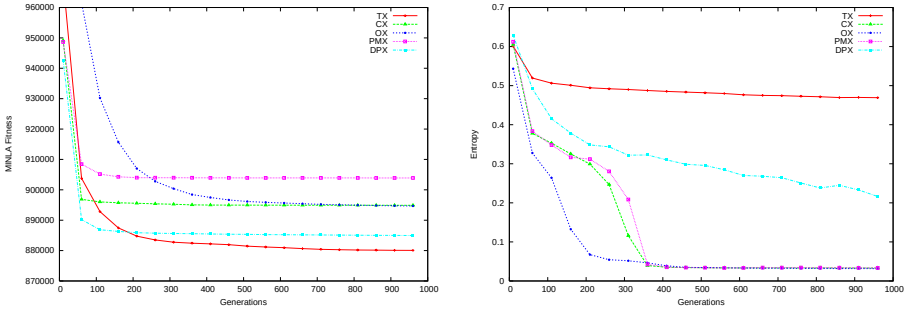
In order to enable a fair comparison all the recombination operators were tested under the same conditions on three representative instances ($randomA1$, $c2y$ and $mesh33x33$) taken from the the Petit's test-suite [16]. The following parameters were used for the MA in this experiment: a) population size $|P| = 100$, b) recombinations per generation $offspring = 50$, c) maximal number of local search iterations $L = 500$, d) maximal number of generations $maxGenerations = 1500$ and e) maximal number of successive failed generations $maxFails = 100$. A relative small number of local search iterations was used, to reduce the strong influence of the local search in the results.

Due to the non-deterministic nature of the algorithm, 20 independent runs were executed for each instance/operator combination. The results of these executions are summarized in Table 1. For each instance/operator combination we present the average population fitness and the average population entropy after 1500 generations. This table shows clearly that the TX operator allows us to obtain better results for the three graphs while conserving also the population diversity. In our experiments we have tested other instances and they provide similar results. This dominance is better illustrated in Fig. 2, where the behavior of the studied operators is presented over the $randomA1$ instance. In Fig. 2(a) the X axis represents the number of generations, while the Y axis indicates the average population fitness. Fig. 2(b) presents the evolution of the population entropy (Y axis) with respect to the number of generations. Observe that the best trade-off between exploration and exploitation is obtained by the TX operator.

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

Table 1. Comparison of memetic recombination operators for MinLA

Operator	randomA1		mesh33x33		c2y	
OX	894603.0	0.031	35054.6	0.031	89088.3	0.033
PMX	916023.3	0.334	35952.3	0.218	89193.0	0.223
DPX	899313.6	0.208	34975.3	0.258	84651.3	0.296
CX	891294.0	0.331	35041.6	0.032	84673.3	0.035
TX	881023.0	0.469	34827.0	0.305	83865.0	0.277



(a) (b)

Fig. 2. Graphs representing the behavior of 5 memetic recombination operators over the *randomA*1 instance. (a) Average population fitness, (b) Population entropy.

4.2 Comparison with the Best Known Results

In the second experiment we have tunned the combination of our MA and the TX operator (MA+TX). Then a performance comparison of the MA+TX procedure with the following heuristics was carried out: SS+SA [16], DT+SA [1], AMG [18] and GH [17]. In this experiment the MA+TX parameters were: a) population size $|P| = 50$, b) recombinations per generation $offspring = 25$, c) maximal number of local search iterations $L = 10000$, d) maximal number of generations $maxGenerations = 10000$ and e) maximal number of successive failed generations $maxFails = 100$.

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 papers. Finally, the last column presents the difference (Δ_C) between the best total edge length found by MA+TX and the previous best known solution reported in the literature.

From Table 2, one observes that MA+TX is able to improve on 4 previous best known solutions and to equal these results in 5 instances. For the other instances, MA+TX did not reach the best reported solution, but its results are very close to the best reported (in average 1.1%). Notice that for some

instances the improvement is important; leading to a significant decrease of the total edge length (Δ_C up to -6108). With respect to the computational effort we have noted that MA+TX, given that it is a memetic algorithm, consumes considerably more computer time than some heuristics especially developed for MinLA such as DT [1], MS [13] and AMG [18].

Table 2. Performance comparison between MA+TX and several state-of-the-art algorithms.

Graph	$\|V\|$	$\|E\|$	SS+SA	DT+SA	AMG	GH	MA+TX	Δ_C
randomA1	1000	4974	869648	884261	888381	878637	868724	-924
randomA2	1000	24738	6536540	6576912	6596081	6550292	6535849	-691
randomA3	1000	49820	14310861	14289214	14303980	14246646	14240538	-6108
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	3808	112
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	363079	5750
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	79420	598
c3y	1327	2844	123548	127065	123514	125654	123521	7
c4y	1366	2915	116140	115222	115131	117539	115204	73
c5y	1202	2557	97791	96956	96899	98483	96962	63
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

5 Conclusions

In an extensive study, several recombination operators, including the new Trajectory Crossover (TX) operator, were compared within a Memetic Algorithm framework. From this comparison we can conclude that the best trade-off between exploration and exploitation is obtained by the TX operator. Furthermore, the performance of our MA+TX 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 [16], DT+SA [1], MS [13] and GH [17]. The results obtained by MA+TX are superior to those presented by the previous proposed evolutionary approach [17], and permit to improve on some previous best known solutions.

There are some issues for future research. For example, to investigate the behavior of MA+TX when it is applied to larger instances, like those proposed by Koren and Harel in [13], in order to study its scalability.

Acknowledgments. This work is supported by the CONACyT Mexico, the "Contrat Plan Etat Région" project COM (2000-2006) as well as the Franco-Mexican Joint Lab in Computer Science LAFMI (2005-2006). The reviewers of the paper are greatly acknowledged for their constructive comments.

References

1. 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.
2. L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the IJCAI*, pages 162–164. Morgan Kaufmann, 1985.
3. J. Diaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.
4. S. Even and Y. Shiloah. NP-completeness of several arrangement problems. Technical Report CS0043, Technion, Israel Institute of Technology, Haifa, Israel, 1975.
5. B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Proc. Of the 1996 IEEE Int. Conf. On Evolutionary Computation*, pages 616–621. IEEE Press, 1996.
6. P. Galinier and J. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
7. M. Garey and D. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
8. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
9. D. E. Goldberg and R. Lingle. Alleles, loci, and the travelling salesman problem. In *Proc. Of an Int. Conference on Genetic Algorithms and their Applications*, pages 154–159. Carnegie Mellon publishers, 1985.
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. 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.
14. P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms and greedy operators for graph bi-partitioning. *Evolutionary Computation*, 8(1):61–91, 2000.
15. I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the travelling salesman problem. In *Proc. Of the 2nd. Int. Conference on Genetic Algorithms*, pages 224–230. Lawrence Erlbaum Associates, 1987.
16. J. Petit. *Layout Problems*. PhD thesis, Universitat Politécnica de Catalunya, 2001.
17. T. Poranen. A genetic hillclimbing algorithm for the optimal linear arrangement problem. Technical report, University of Tampere, Finland, June 2002.
18. I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 2004. in press.
19. D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proc. Of the 3rd Int. Conf. On Genetic Algorithms*, pages 133–140. Morgan Kaufmann, 1989.