Sequencing Cutting Patterns and VLSI Gates by Population Training Algorithms

Alexandre C. M. Oliveira • Luiz A. N. Lorena •

Departamento de Informática, Universidade Federal do Maranhão, Av. Portugueses s/n, Campus do Bacanga, São Luís, Maranhão 65085-580, Brasil

Laboratório Associado de Computação Aplicada, Instituto Nacional de Pesquisas Espaciais, Av Astronautas 1758, Jardim da Granja, São José dos Campos, São Paulo 12227-010, Brasil

acmo@deinf.ufma.br • lorena@lac.inpe.br •

This paper describes a new way to employ problem-specific heuristics to improve evolutionary algorithms: the Population Training Algorithm (PTA). The PTA keeps stored the individual and its best neighbor in the population for a number of generations inversely proportional to the difference between their evaluation. The population is then ranked by a coefficient that contemplates the double evaluation of individuals, in order that, the best individuals by this rank have greater probability to be selected for recombination and mutation operations. Applications are examined for two sequencing problems: the gate matrix layout and the minimization of open stacks. A 2-Opt-like heuristic and other based upon Faggioli and Bentivoglio's greedy procedure are employed as training heuristics and their performance are compared, using instances taken from the literature.

(*Hybrid evolutionary algorithms; Population training; MOSP; GMLP*)

---

1. Introduction

Evolutionary algorithms are efficient to explore a wide search space, converging quickly to local minima. However, their lack of exploiting local information is a well-known drawback to reach a global minima. Heuristics and local search procedures has been encapsulated by evolutionary operators to incorporate knowledge about problems' particularities, driving the evolutionary process to build a population with some desired feature, gaining speed and accuracy. The local search procedures are, in general, applied to only individuals considered elite, avoiding unnecessary objective function calls. For the same reason, problem-specific

heuristics has been employed to analyze and manipulate individuals' structure, improving it without objective function evaluation.

The Constructive Genetic Algorithm (CGA) was proposed in (Lorena and Furtado 2001) to location problems, and applied to timetabling (Lorena and Ribeiro 2001). The CGA a number of new features compared to a traditional genetic algorithm, such as a dynamic-sized population composed of schemata (incomplete solutions) and structures (complete solutions), and the possibility of employing heuristics in structure representation and in the fitness function definitions.

Applications of CGA to sequencing problems, such as the gate matrix layout (Oliveira and Lorena 2002), and the minimization of open stack (Oliveira and Lorena 2002), have employed a 2-Opt-like heuristic to define the fitness function. The individual has a double evaluation based upon the solution that it represents and an estimate of how much it can be improved by the 2-Opt-like heuristic.

Considering such CGA applications, two processes are identified and performed in parallel: a 2-Opt training and a constructive process. The structures and schemata are trained with the 2-Opt-like heuristic, and at the same time, more and more structures are built from the initial population composed only by schemata. These applications were the first to evaluate the individuals based upon a problem-specific heuristic and the obtained results were encouraging. However, some relevant aspects of population training were not studied in previous works, as the flexibility of the algorithm to incorporate other training heuristics and number of objective function calls.

In this work, only the heuristic training process is considered and new features are aggregated to the algorithm, such as a new ranking (without contemplating the constructive process) and evolutionary operators. The employment of two distinct training heuristics is focused and the results are compared. This approach is called population training algorithm (PTA).

This paper is organized as follows. Section 2 presents the guidelines of PTA and Section 3 presents theoretical aspects of the problems at hand. The aspects of modeling, such as representation, evaluation and ranking of individuals are presented in Section 4. Section 5 describes the evolutionary process and how the dynamic-sized population is controlled. Section 6 presents the training heuristics employed in this work and Section 7 presents the operators and the algorithm implemented. A comparison between different training heuristics and approaches in some issues of performance are presented in Section 8.

2. General Aspects of PTA

Typical hybrid evolutionary algorithms work as follows. Whether an individual is the best one inside a delimited search subspace (called neighborhood), it is kept in population. Otherwise, it is replaced as soon as a better solution is obtained. Local search heuristics can be applied on a given individual to evaluate its neighborhood.

In PTA, on the other hand, another way to deal with a better solution obtained by heuristic is proposed: keep both, the individual and its best neighbor, in the population for a number of generations inversely proportional to the difference of evaluation between them. The population is then ranked by a coefficient that contemplates this double evaluation of individuals, in order that, the best individuals by this rank have greater probability to be selected from the population for recombination and mutation operations.

The process works as whether the population was trained by a problem-specific heuristic (called training heuristic), where well-adapted individuals are privileged. The evolutionary process penalizes the ill-adapted individuals, promoting the replacement of them along the generations. The individuals representing local minima (or maxima) solutions (can not be improved by the training heuristic) tend to dominate the population.

Each individual must be evaluated by the objective function, at least, two times, considering one call to evaluate the solution and other call to evaluate the neighborhood around it. For example, in sequencing problems as in (Oliveira and Lorena 2002), a 2-Opt neighborhood is evaluated from 1 up to N objective function calls, where N is the problem length (number of items to be permuted). The best neighbor is reached by changes in the permutation and every new generated permutation is evaluated by the objective function. This procedure works as it was applying a small local search to the whole population and a meaningful number of objective function calls can be required.

In this work, another type of heuristics has been investigated for gate matrix layout and minimization of open stack problems, based upon expertise knowledge about the problems that can decrease the computational effort to evaluate the individual. There is a problem-specific heuristic proposed by Faggioli and Bentivoglio (Faggioli and Bentivoglio 1998) that always generates a single neighbor to be evaluated, considering the similarity among the patterns/gates of the problem. This work presents a comparison between 2-Opt-like heuristic and Faggioli's greedy procedure, being employed as training heuristics.

3. Theoretical Issues of MOSP

Minimization of Open Stacks Problem (MOSP) appears in a variety of industrial sequencing settings, where distinct patterns need to be cut and each one may contain a combination of piece types. For example, consider an industry of woodcut where pieces of different sizes are cut of big foils. Pieces of equal sizes are heaped in a single stack that stays open until the last piece of the same size is cut.

A MOSP consists of determining a sequence of cut patterns that minimizes the maximum number of opened stacks during the cutting process. Typically, this problem is due to limitations of physical space, so that the accumulation of stacks can cause the temporary need of removal of one or other stack, delaying the whole process.

The data for a MOSP are given by an IxJ binary matrix P, representing patterns (rows) and pieces (columns), where $P_{ij} = 1$, if pattern i contains piece j, and $P_{ij} = 0$ otherwise. Patterns are processed sequentially, opening stacks whenever new piece types are processed and closing stacks of pieces that do not have any items else to be cut. The sequence of patterns being processed determines the number of stacks that stays open at same time.

Another binary matrix, here called the open stack matrix Q, can be used to calculate the maximum number of open stacks for a certain pattern permutation. It is derived from the input matrix P, by following rules:

- $Q_{ij} = 1$ if there exists x and $y | \pi(x) \leq i \leq \pi(y)$ and $P_{xj} = P_{yj} = 1$;
- $Q_{ij} = 0$, otherwise;

where $\pi(b)$ is the position of pattern b in the permutation.

Considering matrix Q, the maximum of open stacks (MOS) can be easily computed as:

$$MOS = \max_{i \in I} \left\{ \sum_{j \in J} Q_{ij} \right\} \tag{1}$$

The matrix Q clarifies the stacks that are open (consecutive-ones in the columns) along the cutting of patterns. The Figure 1 shows an example of matrix P, your corresponding matrix Q, and MOS calculated for same example. The Q shows the consecutive-ones property (Golumbic 1980) for columns being applied to P. In each column, one can see when a stack is open (first "1"), and when it is closed (last "1"). Between first and last "1" 's, the stack stays opened ("1" 's sequence).

The sum of "1" 's by rows, computes the number of open stacks when each pattern is processed. For the example of Figure 1, when pattern 1 is cut there are 2 open stacks, then pattern 2 is cut opening 5 stacks, and so on. One can note that, at most, 5 stacks ($MOS = 5$) are needed to process the permutation of patterns $\pi_0 = \{1, 2, 3, 4, 5\}$.

	pieces	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8	Σ
	pattern 1	0	0	1	0	1	0	0	0		0	0	1	0	1	0	0	0	2
P_{ij} =	pattern 2	1	1	0	0	1	1	0	0	Q_{ij} =	1	1	1	0	1	1	0	0	5
	pattern 3	1	0	1	0	0	0	0	0		1	0	1	0	1	0	0	0	3
	pattern 4	0	0	0	1	1	0	0	1		0	0	1	1	1	0	0	1	4
	pattern 5	0	0	1	0	0	0	1	0		0	0	1	0	0	0	1	0	2
																		MOS= max {2,5,3,4,2} =	5

(a) (b)

Figure 1: MOSP instance: a) original problem matrix P; b) corresponding matrix Q

For MOSP, the objective is to find out the optimal permutation of patterns that minimizes the MOS value. The Figure 2 shows Q of the optimal permutation, $\pi_1 = \{5, 3, 1, 2, 4\}$, for the example of Figure 1.

pieces	1	2	3	4	5	6	7	8	Σ
pattern 5	0	0	1	0	0	0	1	0	2
pattern 3	1	0	1	0	0	0	0	0	2
pattern 1	1	0	1	0	1	0	0	0	3
pattern 2	1	1	0	0	1	1	0	0	4
pattern 4	0	0	0	1	1	0	0	1	3
							MOS = max {2,2,3,4,3} =		4

Figure 2: Optimal solution for MOSP instance

Other permutations with $MOS = 4$ can exist, for example $\pi_2 = \{2, 3, 1, 5, 4\}$, but π_1 holds an advantage to the others: the time that the stacks stay open (TOS). The TOS can be calculated by the sum of all "1" 's in Q. It comes from the distance, in the permutation, between the pattern that opens and the pattern that closes each stack. This would be a second objective for MOSP: to close the stacks as soon as possible, allowing that the customer's requests be available.

A more detailed introduction to MOSP can be found in Becceneri (Becceneri 1999) and practical applications in (Yanasse 1996). With respect to complexity of MOSP, some works NP-hardness of MOSP have been published in the last decade. (Andreatta et al., 1989) formulated the cutting sequencing problem as a minimum cut width problem on a hypergraph and showed that it and showed that it is NP-Complete (Andreatta et al. 1989). Recently, presented several aspects of MOSP and other related problems, like the GMLP

(Gate Matrix Layout Problem), including the including the NP-hardness of them (Linhares 2002).

The GMLP is a known NP-hard problem and arises on VLSI design (Möhring 1990), (Kashiwabara and Fujisawa 1979). Its goal is to arrange a set of circuit nodes (gates) in an optimal sequence, such that the layout area is minimized, i.e., it minimizes the number of tracks necessary to cover the gates interconnection. The relationship between MOSP and GMLP resides in the consecutive-ones property:

- a stack is opened at the moment that the first piece of a type is cut and stays opened until the cut of the last piece of this same type, occupying a physical space during this time; at same way,
- a metal link is begun from the leftmost gate in a net and passes by all gates of the circuit until the rightmost gate, occupying a physical space inside of a track.

Concerning input matrix P of MOSP, the consecutive-ones property: occurs in the columns, differently of GMLP that occurs in rows. Figure 3 shows an example of input matrix in GMLP. The corresponding gate matrix is derived by consecutive-ones property applied to rows and, in bottom, one can see the number of track overlaps.

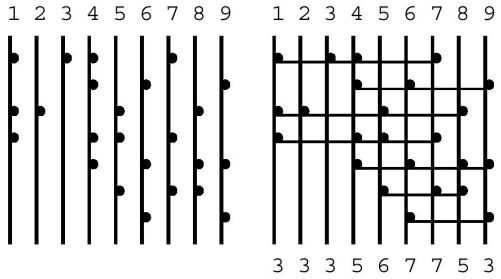

Figure 3: GMLP instance: a) original gate matrix; b) corresponding gate matrix

4. PTA Modeling

The modeling of MOSP and GMLP are quite similar, in general. The main difference between them is the objective function. For this reason, all modeling aspects are treated only for MOSP. More details about GMLP can be found in (Oliveira and Lorena 2002).

A very simple representation is implemented for MOSP and GMLP: a direct alphabet of symbols (natural numbers) represents the pattern (or gate) permutation. Each label is

associated to a row of binary numbers, representing the piece type presence in each pattern. A permutation of rows is called structure and consists of a candidate solution for an MOSP instance. The Figure 4 shows structures for the MOSP instance of Figure 1.

1	0 0 1 0 1 0 0 0	2	1 1 0 0 1 1 0 0
2	1 1 0 0 1 1 0 0	5	0 0 1 0 0 0 1 0
3	1 0 1 0 0 0 0 0	3	1 0 1 0 0 0 0 0
4	0 0 0 1 1 0 0 1	1	0 0 1 0 1 0 0 0
5	0 0 1 0 0 0 1 0	4	0 0 0 1 1 0 0 1

$S_i =$	(1 2 3 4 5)	$S_j =$	(2 5 3 1 4)

Figure 4: Examples of structures (candidate solutions)

In the modeling process, the original problem, MOSP, is transformed in a bi-objective problem that considers how good the structures are with respect the original objective function and other desired features, established by a problem-specific heuristic. The MOSP, being a minimization problem, is modeled as:

$$\begin{cases} \min & \{g(S_k) - f(S_k)\} \\ \min & \{g(S_k)\} \\ & \\ & \textit{Subject to} \qquad g(S_k) \geq f(S_k), \forall S_k \in X \end{cases} \tag{2}$$

All structures, S_k, are evaluated by two fitness functions, f and g, defined on the search space X of all structures that can be obtained by the direct alphabet of symbols. The function g is the original objective function and the function f evaluates certain desirable features, determined by a problem-specific heuristic. Such desirable features could improve the objective function of any structure, but not always that happens. Thus, the interval $g-f$ estimates the difference between "what the structure is" and "what the structure should be".

The bi-objective problem can not be solved directly due to X be unknown or very large, but indirectly, through a genetic algorithm. By this way, all structures $S_k \in X$ are treated as an individual in a population.

For MOSP (or GMLP), the function g reflects the total cost of a given permutation of patterns (or gates). To increase the fitness distinction among the individuals of the population, it is used a formulation that considers the MOS minimization as primary objective and TOS minimization as a secondary one. Therefore, it is defined as $g(S_k) = I \cdot J \cdot MOS(S_k) + TOS(S_k)$ or

$$g(S_k) = I \cdot J \cdot \max_{i \in I} \left\{ \sum_{j \in J} Q_{ij} \right\} + \sum_{i \in I} \sum_{j \in J} Q_{ij} \tag{3}$$

where the product $I \cdot J$ is a weight to reinforce the part of the objective considering the maximum number of open stacks and making it more important than the second part, concerning the time of open stacks.

The fitness function f is defined to drive the evolutionary process to a population trained by a heuristic. A heuristic defines a neighborhood relationship between individuals, delimiting a search subset around them. Typically, a local search procedure can be used to improve the solution, represented by a given individual, S_k, i.e., to find the best solution inside the neighborhood defined by the heuristic H applied to S_k. Thus, function f can be defined by:

$$f(S_k) = \min\{g(S_1), g(S_2), \ldots, g(S_V), g(S_k)\} \tag{4}$$

where $\{S_1, S_2, \ldots, S_V\}$ are a set of structures of the search space, called H-neighborhood of S_k, generated by the training heuristic H, from S_k.

The heuristic H is employed to generate a set of structures (solutions), $\{S_1, S_2, \ldots, S_V\}$, from S_k that are evaluated, aiming to find the best. The best evaluation found, S_v, is assigned to $f(S_k)$. In this work, a 2-Opt like heuristic and a Faggioli's greedy procedure are used as training heuristics. A more detailed description of them is presented later.

Considering, G_{max}, an estimate of the upper bound for all possible values of the function g, the interval I1, $G_{max} - g(S_k)$, gives the distance between structure S_k and the upper bound. In other words, greater the interval I1, better is a structure S_k for a minimization problem. In the population training approach, however, is also considered the adaptation of individual with respect to the training heuristic H, and ill-adapted individuals are penalized by the interval $I2$, $g(S_k) - f(S_k)$. For minimization problems, all individuals are ranked by the following:

$$\delta(S_k) = d \cdot [G_{max} - g(S_k)] - [g(S_k) - f(S_k)] \tag{5}$$

where d is a constant percentage of G_{max}, i.e., a proportionality constant for interval I1. In general, $d = 1/G_{max}(G_{max} > 0)$ and G_{max} can be estimated by sampling or effectively computed, considering the problem instance.

All individuals, S_k, are firstly evaluated by the objective function g, followed by function f, that evaluates the H-neighborhood of S_k. If S_k can not be improved by H, then S_k is well-

adapted to H. Otherwise, the difference $(g(S_k) - f(S_k))$ is subtracted from $d \cdot (G_{max} - g(S_k))$ penalizing the individual. For minimization problems, the well-evaluated individuals have low g evaluation and are well-adapted to H, $g - f \cong 0$.

5. The Evolutionary Process

A dynamic-sized population was implemented and controlled by an adaptive rejection threshold that eliminates the ill-adapted individuals, i.e., structures such that $\alpha \geq \delta(S_k)$.

The adaptive rejection threshold, α, is initialized at the beginning of the process with the rank of the worst individual in population. During the evolutionary process, α is updated with adaptive increments, considering the current range of the rank values in the population, the population size, and the remaining number of generations. The adaptive increment of α is shown in (6):

$$\alpha = \alpha + Step \cdot PS \cdot \frac{(\delta_{bst} - \delta_{wst})}{RG} \tag{6}$$

where δ_{bst} and δ_{wst} are, respectively, the best and the worst rank of structures in current population, PS is the current population size, RG is the remaining number of generations, and $Step$ is an adjustment constant, used to give more or less speed to the evolutionary process.

The values of $Step$ controls the step length of α. Starting from the initial population, new individuals are generated, by recombination and mutation operations, exploring new solutions and their neighborhoods. At the beginning, the population tends to grow up, generally, accepting all new individuals. After some generations, α determines the adaptation values that can be kept in population and the ill-adapted individuals are eliminated. When no improvement is obtained, the population decreases, until becomes empty. Therefore, the parameter $Step$ sets how many generations will happen before the population becomes empty.

6. Training Heuristics

The well-adapted individuals have best ranking and are kept in the population for more generations. To compute the adaptation of individuals, different heuristics can be employed. Each heuristic defines a different neighborhood between structures in the search space.

In general, the heuristic is applied to the individual S_k being evaluated and one or more structures are generated. One can say that such structures are neighbors of S_k. These structures are evaluated with respect to the objective function and the best value is set as $f(S_k)$.

In this work, it is used a 2-Opt-like heuristic and other based upon Faggioli's greedy procedure (Faggioli and Bentivoglio 1998). Both are explained in the next subsections.

6.1 The 2-Opt Heuristic

The 2-Opt-like heuristic is employed to train the population by the fitness function f. Another application to 2-Opt is to perform a local search mutation and is better explained later.

To generate the 2-Opt neighborhood, an iterative process starts, inspecting all possible 2-move changes in the structure. Each 2-move generates a neighbor structure that will be evaluated, looking for the best objective function value. At the end, up to $0.5(N^2 - N)$ neighbor structures are evaluated (N is the number of patterns in problem instance).

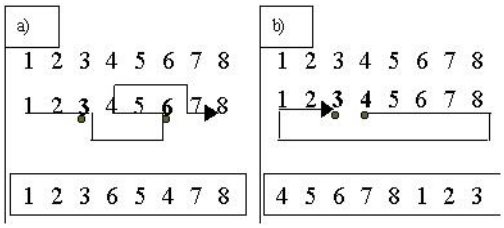

Figure 5: 2-move changes using a) non-consecutive and b) consecutive reference points

An example of 2-move change is showed in Figure 5. The marks in positions of the structures mean reference points to be changed. Non-consecutive references cause the first change type, as showed in Figure 5a. Consecutive points cause the second change type in Figure 5b. For example, inspecting 4 neighbors, from first position in Figure 5, generates 6 pairs of reference points:
$\{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$.

6.2 The Faggioli's Heuristic

The Faggioli's heuristic is also employed to train the population by the fitness function f and is based upon the greedy heuristic procedure described in Faggioli and Bentivoglio's

paper. The basic idea of this heuristic is, from a initial group of patterns, to minimize the differences with respect to a subsequent one, and so on, until one permutation with all patterns is completed.

The initial group of patterns (the first $N/2$ pattern), in a given structure, is accepted as start patterns. The neighborhood is defined as all structures that begin with the start group of patterns and minimize the difference to the subsequent patterns, accordingly with Faggioli's heuristic. Then, the next pattern to be sequenced is chosen from the remaining group, according with a three stage criterion.

At the first stage, the patterns that open as less new stacks as possible are chosen. A stack is opened when the new sequenced pattern contains a piece type that is not yet stacked, i.e., the i_{th} item presents a $0-1$ transition, from previous pattern to next.

At the second stage, the pattern that removes the greatest number of stack is chosen among the patterns previously selected. A stack is removed when the new sequenced pattern ends a piece type that is being stacked, i.e., the i_{th} item presents a $1-0$ transition, from previous pattern to next.

At the last stage, the pattern that continues the production of the greatest number of stacked pieces is chosen among the patterns previously selected. The production continues when the new sequenced pattern contains a piece type that is already stacked, i.e., the i_{th} item presents a $1-1$ transition, from previous pattern to next.

If these three rules lead to more than one pattern to be inserted in sequence, one of them is selected at random. This random decision gives a random nature to neighborhood being built, that is to say, more than one neighbors can exist to a given structure.

7. Evolutionary Operators and Algorithm

The structures in population are kept in descending order, according to the ranking in (5). Thus, well-adapted individuals appear in first places on the population, being privileged for selection, recombination and mutation operations.

7.1 Selection and Recombination

Two structures are selected for recombination. The first is called the base (S_{base}) and is randomly selected out from the first positions in the population. The second structure is called the guide (S_{guide}) and is randomly selected out of the entire population.

In this work, the recombination implemented is a variant of the Order Crossover (OX) called Block Order Crossover (BOX) (Syswerda 1991). The parent(A) and parent(B) are mixed into only one offspring, by copying blocks of both parents, at random. Pieces copied from a parent are not copied from other, keeping the offspring feasible. This crossover tends to be less disruptive than others, as the Partially Matched Crossover (PMX). The Figure 6 shows an example of BOX.

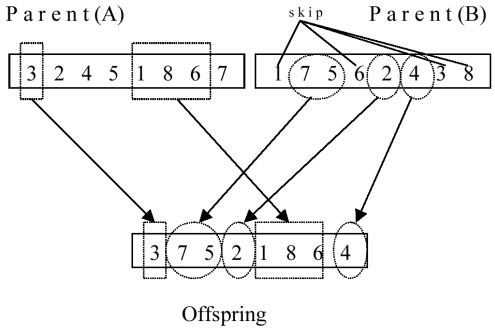

Figure 6: Example of Block Order Crossover (BOX)

7.2 Local Search Mutation

The local search mutation is applied to each new structure generated with a certain probability. This procedure is very important to get intensification moves around a solution. In (Oliveira and Lorena 2002), a stand-alone 2-Opt procedure was also implemented and tested, obtaining good results to small problems, whether compared to others procedures.

The local search mutation considers several better individuals per mutation, exploring a search tree. A neighborhood is inspected in each tree level and the best found structure is held on to be used as starting point to next tree level. Successive neighborhoods are generated and new best structures are held on until a pre-defined maximum number of neighborhoods (J).

To avoid increasing the computational efforts, each neighborhood is limited to a constant number of neighbors (L) around the starting structure. An initial point in structure is chosen at random and an iterative process starts from it, inspecting some 2-move changes in the structure, until the maximum length previously established. Each 2-move generates a neighbor structure that is evaluated, looking for the best in each neighborhood.

The 2-Opt local search mutation is showed in Figure 7. The neighborhood length (L) and the number of neighborhoods (J) are set together with other parameter settings that are

described in next section. Generally, L and J receive the same value, called neighborhood width ($NeighborhoodWidth$).

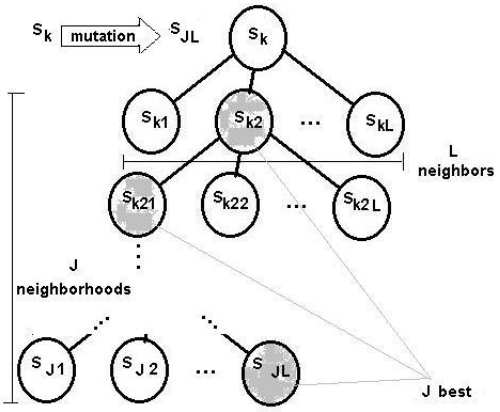

Figure 7: Search tree generated by the 2-Opt local search mutation

7.3 The Algorithm

The algorithm implemented in this work is showed as follows. The end condition can be any of the following events: a) after a given number of generations; b) the emptying of the population.

```
Initialize(P,alpha, t);
while (not END_CONDITION)
    t = t+1;
    Select(P, base, BasePercentage);
    Select(P, guide,WholePopulation);
    New=Recombine(base,guide);
    EvaluateIndividual (new);
    if (rand() < MutationPercetange)
        Mutation(new,NeighborhoodWidth);
    end_if
    EvaluateNeighborhood(new);
    CalculateDelta(new);
    if (delta(new) > alpha)
        UpdateInPopulation(P,new);
    end_if
```

```
    alpha := alpha + AdaptiveIncrement(Step);
    While (exist delta(S) < alpha)
        EliminateFromPopulation(P,S);
    end_while
end;
```

This algorithm is quite intuitive and is based upon traditional genetic algorithms. The following parameter settings are described in next section.

- *BasePercentage*: percentage of individuals considered as possible base individuals;
- *MutationPercentage*: probability of local search mutation;
- *Step*: controls the step length of α;
- *NeighborhoodWidth*: neighborhood width.

The parameter *BasePercentage* controls the diversification of parents in the recombination. *MutationPercentage* controls the number of individuals suffering local search and, consequently, the computational effort during the evolutionary process. The parameter *NeighborhoodWidth* is also associated with the computational effort, during the local search. At last, the parameter *Step* can avoid a premature emptying of the population.

8. Computational Tests

A pool of 300 MOSP instances, taken from Faggioli and Bentivoglio's paper (Faggioli and Bentivoglio 1998), and one GMLP instance taken from literature were chosen for tests. The MOSP instances are grouped by number of patterns $(10, 15, 20, 25, 30, 40)$. Each one of these pattern groups has five piece type subgroups $(10, 20, 30, 40, 50)$ and each piece type subgroup has ten instances with different solutions. The GMLP instance (called *w4*) has 141 gates and 202 nets and is the largest instance found in the literature (Linhares et al. 1999). The main reason to taken these instances is that some best known solutions were found recently and are considered a challenge (Linhares 2002), (Linhares et al. 1999).

The Population Training Algorithm (PTA) for MOSP/GMLP was coded in ANSI C and it was run on Intel AMD (1.33 GHz) hardware. For the computational tests, some PTA

parameters were adjusted. The best results were obtained with *BasePercentage=20%*, *MutationPercentage=20%*, *Step=0.001* and neighborhood width *NeighborhoodWidth=20*. The PTA was initialized with 50 and 100 individuals for MOSP instances and GMLP instance, respectively. Two versions of PTA (*PTAopt* and *PTAfag*), using the 2-Opt-like and Faggioli's greedy heuristics for training, were built and compared to evaluate how different heuristics interfere in the algorithm performance.

In Faggioli and Bentivoglio's work, six methods are presented, and the best two are: a) a tabu search method (TS) based upon an optimized move selection process; and b) a generalized local search method (GLS) that employs the Faggioli's greedy procedure in a simplified tabu search that only accepts improving moves.

In this work, besides the two previously mentioned methods (TS, GLS), another three methods are included for comparison with PTA: c) the Constructive Genetic Algorithm (CGA) (Oliveira and Lorena 2002); d) the Collective method (COL) proposed recently by Linhares (Linhares 2002); and e) a dynamic-sized population genetic algorithm without local search (DPGA).

The original version of CGA employs only 2-Opt-like heuristic as training heuristic, but, in this work, CGA was adapted to work with Faggioli's greedy heuristic too and new simulations were performed with both versions. The CGA's parameter settings are the same found in (Oliveira and Lorena 2002).

The COL method explores distance measures among permutations to drive the search of an algorithm similar to the simulated annealing, where the moves in the search space are based upon exchange in pattern positions. Its results were taken from (Linhares 2002).

The DPGA has the same algorithm structure of PTA, but do not use any local search method. Instead, it uses a mutation that exchanges labels inside structures with certain probability. Besides, DPGA is ranked by traditional fitness, i.e., by the objective function shown in (3). The best DPGA performance was obtained with a population of 1000 individuals (initially) and mutation probability of 5%.

The Table 1 shows the best solution averages obtained by PTA (both versions *PTAfag* and *PTAopt*), COL, TS, GLS, CGA (both versions *CGAfag* and *CGAopt*) and DPGA for each instance group. Only the MOS minimization is compared because the TOS is not considered on the other works. For this test, CGA, DPGA and PTA were run 20 times. Considering the best known solutions, only CGA and PTA have found the best overall average of solutions for the instance groups, i.e., 100% of success. The COL appears with

the second best performance, achieving the best average in 80% of instance groups (24 of 30), DPGA (46% or 14 of 30), TS (37% or 11 of 30), and GLS (33% or 10 of 30) of success rate, respectively.

Table 1: Performance comparison with another approaches per instance groups

I	J	PTA	COL	TS	GLS	CGA	DPGA
10	10	5.5	5.5	5.5	5.5	5.5	5.5
-	20	6.2	6.2	6.2	6.2	6.2	6.2
-	30	6.1	6.1	6.1	6.2	6.1	6.1
-	40	7.7	7.7	7.7	7.7	7.7	7.7
-	50	8.2	8.2	8.2	8.2	8.2	8.2
15	10	6.6	6.6	6.6	6.6	6.6	6.6
-	20	7.2	7.2	7.2	7.5	7.2	7.3
-	30	7.3	7.3	7.4	7.6	7.3	7.4
-	40	7.2	7.2	7.3	7.4	7.2	7.3
-	50	7.4	7.4	7.6	7.6	7.4	7.4
20	10	7.5	7.5	7.7	7.5	7.5	7.5
-	20	8.5	8.5	8.7	8.6	8.5	8.5
-	30	8.8	9.0	9.2	8.9	8.8	9.1
-	40	8.5	8.6	8.6	8.7	8.5	8.5
-	50	7.9	7.9	8.0	8.2	7.9	8.0
25	10	8.0	8.0	8.0	8.0	8.0	8.0
-	20	9.8	9.8	9.8	9.9	9.8	10.0
-	30	10.5	10.6	10.7	10.6	10.5	10.5
-	40	10.3	10.4	10.7	10.6	10.3	10.5
-	50	10.0	10.0	10.1	10.2	10.0	10.1
30	10	7.8	7.8	7.8	7.8	7.8	7.8
-	20	11.1	11.2	11.2	11.2	11.1	11.2
-	30	12.2	12.2	12.6	12.2	12.2	12.4
-	40	12.1	12.1	12.6	12.4	12.1	12.5
-	50	11.2	11.2	12.0	11.8	11.2	11.3
40	10	8.4	8.4	8.4	8.4	8.4	8.4
-	20	13.0	13.0	13.1	13.1	13.0	13.3
-	30	14.5	14.5	14.7	14.6	14.5	14.7
-	40	14.9	15.0	15.3	15.3	14.9	15.1
-	50	14.6	14.6	15.3	14.9	14.6	15.0

The MOSP instances *p2040n6*, *p2540n3*, and *p4050n7* are instances of the groups 20x40, 25x40, and 40x50 and were considered the hardest instances to solve. The success rate for them is about 50% or less and the other approaches were not able to find their solutions. These MOSP instances and the GMLP instance *w4* were chosen for a more detailed performance analysis.

The Table 2 shows the comparison between *PTAfag*, *PTAopt*, *CGAfag* and *CGAopt* in 20 trials (except for *w4*, with 10 trials), considering the found average of MOS (AS), the success rate (SR), the average of objective function calls (FC) and the average of running time (RT) in seconds. FC and RT were measured only for trials that reached success, i.e, for cases in which the best known solution was found. The description of the instance, *IxJ*, means I patterns (or gates), J pieces (nets).

Observing Table 2, *PTAopt* has obtained the best AS and SR on the MOSP instances (*p2040n6*, *p2540n3*, and *p4050n7*). Considering these same instances, *PTAfag* and *CGAopt*

Table 2: Full comparison between PTA and CGA employing 2-Opt-like and Faggioli-Bentivoglio's greedy procedure as training heuristics

	PTA(opt)				PTA(fag)			
Instances	AS	SR (%)	FC (10^6)	RT (s)	AS	SR (%)	FC (10^6)	RT (s)
p2040n6	8,7	30	0,578	20,0	8,7	30	0,405	14,3
p2540n3	10,7	30	0,414	21,7	10,8	20	0,708	30,0
p4050n7	14,7	35	1,052	83,1	14,8	25	1,190	94,6
w4	28,6	10	8,488	6.935,5	28,3	20	9,330	7.598,0
	CGA(opt)				CGA(fag)			
Instances	AS	SR (%)	FC (10^6)	RT (s)	AS	SR (%)	FC (10^6)	RT (s)
p2040n6	8,9	10	0,322	11,0	8,9	10	0,303	10,0
p2540n3	10,7	30	0,747	32,3	10,9	10	1,019	36,0
p4050n7	14,7	30	0,749	52,8	14,8	20	0,788	62,3
w4	28,0	70	6,537	6.592,1	29,0	10	10,240	8.264,5

has obtained similar AS and SR results. It can not be distinguished a winner between them, except by the FC. In fact, *CGAopt* has called the objective function lesser than any other version for all instances taken for this test (MOSP and GMLP).

Despite the *CGAopt* to have reached good results, even the best for some cases, the *CGAfag* has failed in finding good results and has obtained the worst overall performance. On the other hand, *CGAopt* has pointed out the best AS, SR, FC and RT with respect to GMLP instance *w4*. The constructive process and 2-Opt-like training heuristic seems to be an efficient association for this instance. Faggioli and Bentivoglio's greedy procedure needs information to evaluate the solution neighborhood and CGA population is composed initially only of schemata (incomplete solutions). This can explain the reason for this performance gap between them.

Considering the training heuristic employed by PTA, one can note their overall performance are slightly equivalent with respect to AS and SR. Once more, a overall winner heuristic can not be found. However, as one can see in convergence trail experiment (seen at following), *PTAfag* has presented an faster approaching to best solution.

Despite the Faggioli and Bentivoglio's greedy procedure seemingly could performe less function calls than 2-Opt-like heuristic, this can not be observed in FC. It was expected a superior FC for versions employing 2-Opt. This fact can be explained perhaps by the mutation procedure: the mutation would dominate the number of function calls and the training heuristic was not relevant for FC. Another possibility is that 2-Opt-like training heuristic would improve the algorithm performance so that it could compensate its computational cost.

The running time (RT) presented by all versions was coherent with the FC. One can

see that the rate RT/FC is basically the same for all, i.e., $RT/FC = 0.0008$ seconds per function call (s/fc), except by the *CGAopt* with $RT/FC = 0.0010 s/fc$.

The convergence trail of PTA, CGA and DPGA can be observed in Figure 8. The graphic shows the average of the best found solution sampled at each 10^5 objective function evaluations for instance *w4* in 10 trials. The convergence trail was split off (Figure 8a and Figure 8b) for clarifying the experiment.

DPGA approaches very fast to the best known solution (27 tracks), but was not able to reach it, probably, due to diversity loss. *PTAfag* slowly approaches to 27 tracks, reaching a good average solution at last, much better than *PTAopt*. However, as one can observe in Figure 8, CGA has the best behavior, with *CGAopt*, and the worst too, with *CGAfag*, among all the population training approaches.

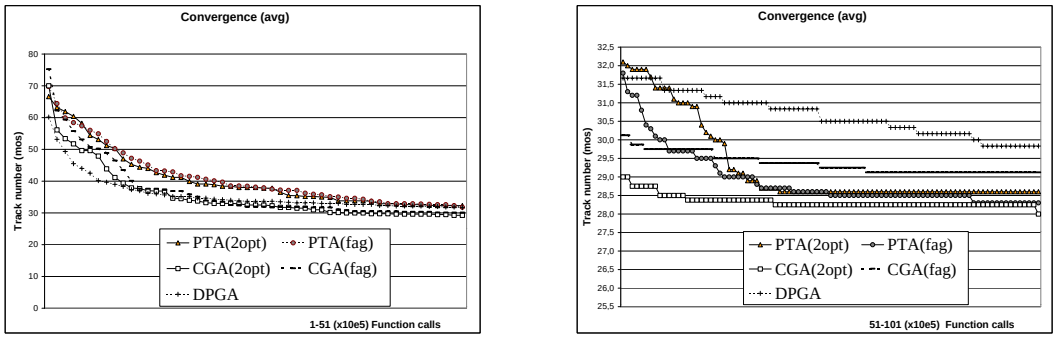

Figure 8: Convergence trail for instance *w4*

9. Conclusion

This paper describes a new way to employ problem-specific heuristics to improve evolutionary algorithms: the Population Training Algorithm (PTA). The PTA keeps stored the individual and its best neighbor in the population for a number of generations inversely proportional to the difference between their evaluation. To define the individual's neighborhood, different heuristics can be employed, and the best individuals are called well-adapted to the heuristic.

Two versions of PTA (*PTAopt* and *PTAfag*), using the 2-Opt-like heuristic and Faggioli and Bentivoglio's greedy procedure for training, were built and compared to evaluate how different heuristics interfere in the algorithm performance. Computational tests has showed

that their overall performance are slightly equivalent, but for a GMLP instance *w4*, *PTAfag* has presented a better convergence.

PTA was also compared with other approaches, taken from the literature, including another algorithm that employs training heuristic: the Constructive Genetic Algorithm (CGA). The CGA combines two processes performed in parallel: a heuristic training and a constructive process. In the constructive process, well-adapted solutions are built from an initial population of schemata.

PTA and CGA were the only ones to find all best known solutions of a pool of 300 MOSP/GMLP instances. PTA has presented the best average solution and success rate for MOSP instances, and CGA for the biggest GMLP instance (*w4*).

Acknowledgments

The first author acknowledges the Programa Institucional de Capacitação Docente e Técnica -PICDT/CAPES for financial support. The second author acknowledges Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (proc. 300837/89-5) and Fundação para o Amparo a Pesquisa no Estado de São Paulo - FAPESP (proc. 99/06954-7) for partial financial support. The authors acknowledge Professors Faggioli and Bentivoglio, that made available the set of MOSP instances used in this paper.

References

Andreatta, G., A. Basso, A. Caumo, L. Deserti. 1989. Un problema min cutwidth generalizzato e sue applicazionni ad un FMS. *Atti delle giornate di lavoro AIRO*, 1-17.

Becceneri, J.C. 1999. O problema de sequenciamento de padrões para minimização do número máximo de pilhas abertas em ambientes de corte industriais. Doctoral Thesis, Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, Brazil.

Faggioli, E., C.A. Bentivoglio. 1998. Heuristic and exact methods for the cutting sequencing problem. *European Journal of Operational Research*, **110** 564-575.

Golumbic, M. 1980. Algorithmic Graph Theory and Perfect Graphs. *Academic Press*. New York.

Kashiwabara, T., T. Fujisawa. 1979. NP-Completeness of the problem of finding a minimum clique number interval graph containing a given graph as a subgraph, *In Proc. Symposium of Circuits and Systems*.

Linhares, A., H. Yanasse, J.R.A. Torreão. 1999. Linear gate assignment: a fast statistical mechanics approach, *IEEE Transactions on Computer-Aided Designed of Integrated Circuits and Systems*, **18(12)** 1750-1758.

Linhares, A. 2002. Industrial Pattern Sequencing Problems: Some Complexity Results And New Local Search Models. Doctoral Thesis, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, SP, Brazil.

Lorena, L.A.N., J.C. Furtado. 2001. Constructive genetic algorithm for clustering problems. *Evolutionary Computation*, **9(3)** 309-327.

Möhring, R. 1990. Graph problems related to gate matrix layout and PLA folding, *Computing*, **7** 17-51.

Oliveira, A.C.M., L.A.N. Lorena. 2002. A Constructive Genetic Algorithm for Gate Matrix Layout Problems. *IEEE Transaction on Computer-Aided Designed of Integrated Circuits and Systems*, **21(8)** 969-974.

Oliveira, A.C.M., L.A.N. Lorena. 2002. 2-Opt Population Training for Minimization of Open Stack Problem. *Advances in Artificial Intelligence. LNAI 2507. Springer*. XVI Brazilian Symposium on Artificial Intelligence. Guilherme Bittencourt e Geber L. Ramalho (Eds). Porto de Galinhas, PE, Brazil. 313-323.

Ribeiro Filho, G., L.A.N. Lorena. 2001. A Constructive Evolutionary Approach to School Timetabling. *Applications of Evolutionary Computing, LNCS 2037. Springer*. Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H.(Eds.) 130-139.

Syswerda, G. 1991. Schedule optimization using genetic algorithms. *Handbook of Genetic Algorithms*,Van Nostrand Reinhold, New York. 332-349.

Yanasse, H.H. 1996. Minimization of open orders-polynomial algorithms for some special cases. *Pesquisa Operacional*, **16** 1-26.