2-Opt Population Training for Minimization of Open Stack Problem

Alexandre César Muniz de Oliveira
DEINF/UFMA
Av. dos Portugueses, s/n, Campus do Bacanga, São Luíz MA, Brazil.
acmo@deinf.ufma.br

Luiz Antonio Nogueira Lorena
LAC/INPE
Caixa Postal 515
12.201-970 São José dos Campos - SP Brazil
lorena@lac.inpe.br

Abstract. This paper describes an application of a Constructive Genetic Algorithm (CGA) to the Minimization Open Stack Problem (MOSP). The MOSP happens in a production system scenario, and consists of determining a sequence of cut patterns that minimizes the maximum number of opened stacks during the cutting process. The CGA has a number of new features compared to a traditional genetic algorithm, as a population of dynamic size composed of schemata and structures that is trained with respect to some problem specific heuristic. The application of CGA to MOSP uses a 2-Opt like heuristic to define the fitness functions and the mutation operator. Computational tests are presented using available instances taken from the literature.

1 Introduction

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 wood cut 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 the 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.

This paper describes the application of a Constructive Genetic Algorithm (CGA) to MOSP. The CGA was recently proposed by Lorena and Furtado [1] and applied to Timetabling and Gate Matrix Layout Problems [2-3], and differs from messy-GAs [4]-[6], basically, for evaluating schemata directly. It also has a number of new features compared to a traditional genetic algorithm. These include a population of dynamic size composed of schemata and structures, and the possibility of using heuristics in structure representation and in the fitness function definitions.

The CGA evolves a population, initially formed only by schemata, to a population of well-adapted structures (schemata instantiation) and schemata. Well-adapted structures are solutions, which cannot be improved using a specific problem heuristic. In this work, it is used a 2-Opt like heuristic to train the population of structures and schemata.

The CGA application can be divided in two phases, the constructive and the optimal: a) the constructive phase is used to build a population of quality solutions, composed of well-adapted schemata and structures, through operators as selection, recombination and specific heuristics; and b) the optimal phase is conducted simultaneously and transforms the optimization objectives of the original problem on an interval minimization problem that evaluates schemata and structures in a common way. In this paper, CGA is applied to MOSP and further conjectures are approached, as the performance of 2-Opt heuristic that is used to define the fitness functions and the mutation operator.

This paper is organized as follows. Section 2 presents theoretical aspects of MOSP. Section 3 presents the aspects of modeling for schema and structure representations and the consideration of the MOSP as a bi-objective optimization problem. Section 4 describes the some CGA operators, namely, selection, recombination and mutation. Section 4 shows computational results using instances taken from the literature.

2 Theoretical Issues of MOSP

The data for a MOSP are given by an IxJ binary matrix P_{ij}, representing patterns (rows) and pieces (columns), where $P_{ij}=1$, if pattern i contains piece j, and $P_{ij}=0$ otherwise. Each pattern is processed by your time, piece by piece, opening stacks (when a new piece type is cut) and closing stacks (when all items of a same that piece type were cut). The sequence of patterns being processed determines the number of stacks that stays open at same time.

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

- qij = 1 if there exists x and y | $\pi(x) \leq i \leq \pi(y)$ and pxj = pyj = 1
- 0, otherwise; where π (b) is the position of pattern b in the permutation.

Considering matrix Q_{ij}, the maximum of open stacks (MOS) can be easily computed as:

$$MOS = \max_{i \in \{1,\ldots,I\}} \sum_{j=1}^{J} q_{ij} \quad (1)$$

The matrix Q_{ij} clarifies the stacks that are open (consecutive-ones in the columns) along the cutting of patterns. The Table 1 shows an example of matrix P_{ij}, your corresponding matrix Q_{ij}, and MOS calculated for same example. The Q_{ij} shows the consecutive-ones property [7] for columns being applied to P_{ij}. 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 Table 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 need to process the permutation of patterns ρ_0={1, 2, 3, 4, 5}.

Table 1. Example of matrices P ij and Qij

	pieces	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8	∑
	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

In MOSP, the objective is to find out the optimal permutation of patterns that minimizes the MOS value. The Table 2 shows Q_{ij} of the optimal permutation, ρ_1={5,3,1,2,4}, for the example of Table 1.

Table 2. Optimal solution

pieces	1	2	3	4	5	6	7	8	∑
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

Other permutations with MOS=4 can exist, for example ρ_2={2,3,1,5,4}, but ρ_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_{ij}. 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 in 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 [8] and practical applications in [9]. With respect to complexity of MOSP, some works approaching the NP-hardness of MOSP have been published in the last decade. Andreatta et al. (1989) formulated the cutting sequencing problem as a minimum cutwidth problem on a hypergraph and showed that it is NP-Complete [10]. Recently, Linhares (2002) presented several aspects of MOSP and other related problems, like the GMLP (Gate Matrix Layout Problem), including the NP-hardness of them [11].

The GMLP is a known NP-hard problem and arises on VLSI design [12-13]. 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) a stack is open at moment that the first piece of a type is cut and stays open until the cut of the last piece of this same type, occupying a

physical space during this time; at same way, b) a metal link is begun from the leftmost gate requiring connection in a net and passes by all gates in circuit until the rightmost gate requiring connection, occupying a physical space inside of a track. Concerning input matrix P_{ij} of MOSP, this property occurs in columns, differently of GMLP that occurs in rows.

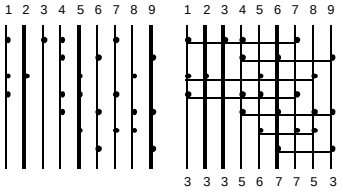

Fig. 1. Example of an input matrix in GMLP. a) Original gate matrix; b) Gate matrix derived by consecutive-ones property applied on rows and, in bottom, the number of track overlaps.

3 CGA Modeling

Very simple structure and schema representations are implemented to the MOSP. A direct alphabet of symbols (natural numbers) represents the pattern permutation and each pattern is associated to a row of binary numbers, representing the piece type presence in each pattern. The symbol # is used to express indetermination (# - do not care) on schemata. Fig.2 shows the representation for the MOSP instance of Table 1, and examples of structures and a schema. The symbols '?' mean there is no information in this row, once the pattern number is an indetermination '#'.

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	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	4	0 0 0 1 1 0 0 1
s_i =	(1 2 3 4 5)	s_j =	(2 5 3 1 4)	s_k =	(# 5 # # 4)

Fig. 2. Examples of structures (S_i and S_j) and schema (S_k)

To attain the objective of evaluating schemata and structures in a common way, two fitness functions are defined on the space X of all schemata and structures that can be obtained this representation. The MOSP is modeled as the following Bi-objective Optimization Problem (BOP):

$$\begin{aligned} &Min && \{g(s_k) - f(s_k)\} \\ &Max && g(s_k) \\ &Subject\ to && g(s_k) \geq f(s_k) \quad \forall\ s_k \in C \end{aligned} \tag{2}$$

Function *g* is the fitness function that reflects the total cost of a given permutation of patterns. To increase the fitness differentiation among the individuals of the population, it is used in the 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

$$g(s_k) = I \cdot J \cdot \max_{i \in \{1,\ldots,I\}} \sum_{j=1}^{J} q_{ij} + \sum_{i=1}^{I} \sum_{j=1}^{J} q_{ij} \quad (3)$$

where the I·J product is a weight to reinforce the part of the objective considering the maximum number of open stacks and to make it proportional to the second part of the objective concerning the time of open stacks. If s_k is schema, the non-defined columns (# label) are bypassed. It seems as these columns do not exist and the Q_{ij} matrix used to compute $g(s_k)$ contains only columns with information. In the example of Fig 2, the MOS is max{?, 2, ?, ?, 3} = 3 and the TOS is sum{0+2+0+0+3} =5.

The other fitness function *f* is defined to drive the evolutionary process to a population trained by a heuristic. The chosen heuristic is the 2-Opt neighborhood. Thus, function *f* is defined by:

$$f(s_k) = g(s_v), s_v \in \{s_1, s_2, \ldots, s_V\} \subseteq \varphi^{2-Opt}, g(s_v) \leq g(s_k) \quad (4)$$

where $\varphi^{2\text{-Opt}}$ is a 2-Opt neighborhood of structure or schema s_k.

By definition, *f* and *g* are applied to structures and schemata, just differing in the amount of information and consequently in the values associated to them. More information means larger values. In this way, the *g* maximization objective in *BOP* drives the constructive phase of the CGA aiming that schemata will be filled up to structures.

4 Evolution Process

The *BOP* defined above is not directly considered as the set *X* is not completely available. Alternatively is considered an evolution process to attain the objectives (*interval minimization* and *g maximization*) of the *BOP*. At the beginning of the process, two *expected values* are given to these objectives:

- g maximization: a non-negative real number $g_{max} > \max_{s \in X}\{g(s)\}$ that is an upper bound on the objective value;
- interval minimization: an interval length dg_{max}, obtained from g_{max} considering a real number $0<d\leq 1$.

The evolution process is then conducted considering an adaptive rejection threshold, which contemplates both objectives in *BOP*. Given a parameter $a \geq 0$, the expression

$$g(s_k) - f(s_k) \geq d \cdot g_{max} - a \cdot d \cdot [g_{max} - g(s_k)] \quad (5)$$

presents a condition for rejection from the current population of a schema or structure s_k. The right hand side of (5) is the threshold, composed of the expected value to the interval minimization $d \cdot g_{max}$, and the measure $g_{max} - g(s_k)$, that shows the difference of $g(s_k)$ and g_{max} evaluations.

Expression (5) can be examined varying the value of a. For $a=0$, both schemata and structures are evaluated by the difference $g-f$ (first objective of *BOP*). When a increases, schemata are most penalized than structures by the difference $g_{max} - g$ (second objective of *BOP*).

Parameter a is related to time in the evolution process. Considering that the good schemata need to be preserved for recombination, the *evolution parameter* a starts from 0, and then increases slowly, in small time intervals, from generation to generation. The population at the evolution time a, denoted by P_a, is dynamic in size accordingly the value of the adaptive parameter a, and can be emptied during the process. The parameter a is now isolated in expression (6), thus yielding the following expression and corresponding rank to s_k:

$$\alpha \geq \frac{d \cdot g_{\max} - [g(s_k) - f(s_k)]}{d[g_{\max} - g(s_k)]} = \delta(s_k). \quad (6)$$

At the time they are created, structures and/or schemata receive their corresponding *rank value* $d(s_k)$. These ranks are compared with the current evolution parameter a. The higher the value of $d(s_k)$, and better is the structure or schema to the *BOP*, and they also have more surviving and recombination time.

For the MOSP, the overall bound g_{max} is obtained at the beginning of the CGA application, by generating a random structure and making g_{max} receive the g evaluation for that structure. In order to ensure that g_{max} is always an upper bound, after recombination, each new structure generated s_{new} is rejected if $g_{max} £ g(s_{new})$.

4.1 Selection and Recombination

The structures and schemata in population P_a are maintained in ascending order, according to the key:

$$\Delta(s_k) = (1 + \frac{g(s_k) - f(s_k)}{g(s_k)}) \cdot \frac{1}{h} \quad (7)$$

where η is the number of genes containing information (not #). Thus, well-adapted individuals (small $g(s_k) - f(s_k)$) with more genetic information (higher h) appear in first order places on the population.

Two structures and/or schemata are selected for recombination. The first is called the *base* (s_{base}) and is randomly selected out of the first positions in P_a, and in general it is a good structure or a good schema. The second structure or schema is called the *guide* (s_{guide}) and is randomly selected out of the total population. The objective of the s_{guide} selection is the conduction of a guided modification on s_{base}.

In the recombination operation, the current labels in corresponding positions are compared. Let s_{new} be the new structure or schema (offspring) after recombination. Structure or schema s_{new} is obtained by applying only one of the following operations:

```
{ Recombination }
For i from 1 to individual length
1) if   s_BASE(i)=# and s_GUIDE(i)= #
       set s_NEW(i)= #

2) if   s_BASE(i) = # and s_GUIDE(i)<> #
       if s_GUIDE(i)is not in s_NEW
          set s_NEW(i)= s_GUIDE(i)
       else  set s_NEW(i)= #

3) if   s_BASE(i)<># and s_GUIDE(i)= #
       if s_BASE(i)is not in s_NEW
          set s_NEW(i)= s_BASE(i)
       else set s_NEW(i)= #

4) if   s_BASE(i)<> # and s_GUIDE(i)<> #
       if s_BASE(i)is not in s_NEW
          set s_NEW(i)= s_BASE(i)
       else
          if s_GUIDE(i)is not in s_NEW
          set  s_NEW(i)= s_GUIDE
          else set s_NEW(i) = #
```

Observe that s_{base} is a privileged individual to compose s_{new}, but it is not totally predominant. There is a small probability of the s_{guide} gene information to be used instead of s_{base} one. More detailed information about CGA features to permutation problems can be found in [3].

4.2 The 2-Opt Heuristic

The 2-Opt like heuristic is used to train the population by the fitness function f. The well-adapted individuals have better ranking and are maintained in the population for more generations. Another application to 2-Opt is to run a local search mutation that is always applied to structures (not to schemata).

To avoid the increasing of computational efforts, only a constant number of neighbors around the structure is inspected, looking for the best. The neighbors are generated by all the 2-move changes in a constant length part of the structure. An initial position is chosen at random and an iterative process starts from it, inspecting all possible 2-move changes in the structure until a maximum length previously established. Each 2-move generates a neighbor structure that will be evaluated and the best one will be hold on.

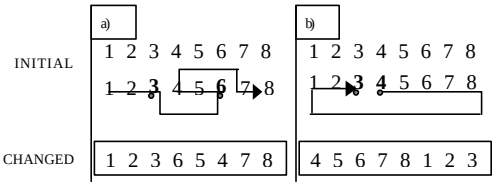

Fig. 3. Examples of one-move in 2-Opt neighborhood; a) non-consecutive reference points change; b) consecutive reference

The example of 2-move change is showed in Fig.3. The marks in positions of the structures mean reference points to be changed. Non-consecutive references cause the first change type, as showed in Fig.3a. Consecutive points cause the second change type in Fig.3b. For example, inspecting 4 neighbors, from first position in Fig.3, generates 6 pairs of reference points: {(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)}, i.e., $0.5*nw*(nw-1)$ pairs, where nw is the neighborhood width and, together with other parameter settings, will be described in next section.

5 Computational Tests

The CGA for MOSP was coded in *ANSI C* and it was run on Intel Pentium II (266Mhz) hardware. For the computational tests, some CGA parameters were adjusted. The d parameter was set to 0.15 (usually between 0.10 and 0.20 values, for other applications [1-2]). This configures the interval $d \times g_{max}$, establishing the survival time of each individual, once the expected d values are proportional to this interval. The e was set to 0.001 and also contributes to the higher survival time of each individual in P_a. These parameters avoid the premature termination with an empty population.

Each schema of the initial population received 50% of # genes (indetermination percentage), and 20% of population (the first individuals ranked by expression 7) were considered base individuals for base-guide selection, determining a small degree of diversification in selection process.

Local search mutation rate was fixed in 100%, which means a constant improvement of individuals. The number of individuals initially generated was proportional to problem length (at least the number of patterns). Other important parameter to be tuned is the neighborhood width (*nw*) to each local search mutation. After some simulations the better results arise for $nw = 20$. The ideal situation would be to use greater values for *nw*, but this would turn the mutation very slow.

The CGA was initially applied to 300 instances taken from the paper of Fraggioli and Bentivoglio [14]. These 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.

In Fraggioli and Bentivoglio's work are presented six solution methods, and the three best are: a) an implicit enumeration method (OPT) that enhances the implicit search procedure of Yuen and Richardson [15], and is used to verify the optimality of the found solutions; b) a tabu search method (TS) based on an optimized move selection process; and c) a generalized local search method (GLS) that works by employing multiple applications of a simplified tabu search that only accepts improving moves. In this work, besides the three previously mentioned methods (OPT, TS, GLS), another two solution methods are included for comparison with CGA: a) the 2-Opt local search heuristic (2-Opt); and b) the collective method (COL) proposed recently by Linhares [11].

The 2-Opt method employs the same heuristic used to train the population in CGA. Initially, a static population of 20 structures is randomly generated, and 2-Opt is applied for each one of them until no more improvement be found. The best solution is held. The 2-Opt parameter *nw* (neighborhood width) is set to maximum size, i.e., the number of patterns of the problem. This exhaustive local search demands a significant computational effort and the running time for large problems (above 100 patterns) is prohibitive.

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 on exchange in pattern positions.

The Table 3 shows the solution averages obtained by OPT, COL, TS, GLS, CGA and 2-Opt for each instance group. Only the MOS minimization is compared because

the TOS is not considered on the other works. The columns I and J refer to numbers of patterns and piece types of each instance group, respectively. The entries emphasized in gray are better than the reported OPT optimum values. Observe that although claimed to be optimal in [14] some entries in OPT column (instances 15x30, 15x40, 15x50, and 40x40) have higher values than, at least, one of these methods: COL, CGA and 2-Opt . This may appear to be a contradiction but these are the new best bounds.

Considering these new best-known solutions, the CGA 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 87% of instance groups (26 of 30), followed by 2-Opt (73% or 22 of 30), TS (40% or 12 of 30) and GLS (33% or 10 of 30) of success rate, respectively.

Table 3. Solution averages obtained by OPT, COL, TS, GLS, CGA and 2-Opt

I	J	OPT	COL	TS	GLS	CGA	2-Opt	I	J	OPT	COL	TS	GLS	CGA	2-Opt
10	10	5.5	5.5	5.5	5.5	5.5	5.5	25	10	8.0	8.0	8.0	8.0	8.0	8.0
-	20	6.2	6.2	6.2	6.2	6.2	6.2	-	20	9.8	9.8	9.8	9.9	9.8	9.8
-	30	6.1	6.1	6.1	6.2	6.1	6.1	-	30	10.5	10.6	10.7	10.6	10.5	10.5
-	40	7.7	7.7	7.7	7.7	7.7	7.7	-	40	10.4	10.4	10.7	10.6	10.4	10.5
-	50	8.2	8.2	8.2	8.2	8.2	8.2	-	50	10.0	10.0	10.1	10.2	10.0	10.0
15	10	6.6	6.6	6.6	6.6	6.6	6.6	30	10	7.8	7.8	7.8	7.8	7.8	7.8
-	20	7.2	7.2	7.2	7.5	7.2	7.2	-	20	11.1	11.2	11.2	11.2	11.1	11.1
-	30	7.4	7.3	7.4	7.6	7.3	7.6	-	30	12.2	12.2	12.6	12.2	12.2	12.2
-	40	7.3	7.2	7.3	7.4	7.2	7.3	-	40	12.1	12.1	12.6	12.4	12.1	12.2
-	50	7.6	7.4	7.6	7.6	7.4	7.4	-	50	11.2	11.2	12.0	11.8	11.2	11.2
20	10	7.5	7.5	7.7	7.5	7.5	7.5	40	10	8.4	8.4	8.4	8.4	8.4	8.4
-	20	8.5	8.5	8.7	8.6	8.5	8.5	-	20	13.0	13.0	13.1	13.1	13.0	13.0
-	30	8.8	9.0	9.2	8.9	8.8	8.9	-	30	14.5	14.5	14.7	14.6	14.5	14.5
-	40	8.6	8.6	8.6	8.7	8.6	8.6	-	40	15.0	15.0	15.3	15.3	14.9	15.0
-	50	7.9	7.9	8.0	8.2	7.9	8.0	-	50	14.6	14.6	15.3	14.9	14.6	14.9

The comparison between CGA and 2-Opt procedure is meaningful, once CGA employs 2-Opt heuristic for fitness definition and local search mutation. The difference between them is the genetic constructive process that exists behind CGA. Selection, recombination and ranking contribute to the construction of well-adapted structures from an initial population of schemata. All these features seem become CGA more robust than other non-population approaches, like COL and 2-Opt.

One can also suppose that 2-Opt could achieve all best solution averages after several trials. However, Table 3 have showed that the 2-Opt is not to be able to find all best solutions. Besides, 2-Opt turns to be prohibitive for large scale instances (above 100 patterns). This can be best verified by the following experiment. The 2-Opt was applied to an instance of another problem type, the GMLP (Gate Matrix Layout Problem), already mentioned in this paper (see section 2). There is a well-known GMLP instance (namely w4) with 141 gates and 202 nets. This is equivalent to a MOSP instance of 141 patterns of 202 piece types. The 2-Opt procedure was run 10 times for w4 instance and did not achieve the best-known solution (27 tracks). The solution 29 was found after 198 minutes. The CGA reach the 27 tracks in 30% of trials and 87 minutes (average time)[3].

6. Conclusion

This work describes an application of the Constructive Genetic Algorithm (CGA) to Minimization of Open Stack Problems (MOSP). The CGA adapted to work with MOSP uses a 2-Opt heuristic as local search mutation and on definition of the two fitness functions (f and g). The algorithm constructs a population of well-adapted structures trained by the 2-Opt heuristic.

Regarding the computational tests, the CGA reached all the best-known results for instances taken from the literature and presented the best results in comparison to other methods. It also appear to be more robust than the standalone application of procedure 2-Opt.

References

1. L. A. N. Lorena, and J. C. Furtado, "Constructive genetic algorithm for clustering problems," Evolutionary Computation 9(3): 309-327, 2001.
2. G. Ribeiro Filho, and L. A. N. Lorena, "A Constructive Evolutionary Approach to School Timetabling," In Applications of Evolutionary Computing, Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H., (Eds.) - Springer Lecture Notes in Computer Science vol. 2037, pp. 130-139 - 2001
3. A. C. M. Oliveira and L. A. N. Lorena, "A Constructive Genetic Algorithm for Gate Matrix Layout Problems". Accepted to IEEE Transaction on Computer-Aided Designed of Integrated Circuits and Systems. 2002.
4. D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: motivation, analysis, and first results," Complex Systems v. 3: p. 493-530, 1989.
5. D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, accurate optimization of difficult problems using fast messy genetic algorithms," IlliGAL Report No. 93004, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1993.
6. H. Kargupta, "Search, polynomial complexity, and the fast messy genetic algorithm," Ph.D. thesis, IlliGAL Report No. 95008, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1995.
7. M. Golumbic, "Algorithmic Graph Theory and Perfect Graphs". Academic Press, New York. 1980.
8. J.C. Becceneri,. "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, Brazil, 1999.
9. H.H. Yanasse, "Minimization of open orders -polynomial algorithms for some special cases." Pesquisa Operacional, v.16, p.1-26, 1996.
10. G. Andreatta, A. Basso, A. Caumo, and L. Deserti. "Un problema min cutwidth generalizzato e sue applicazionni ad un FMS. Atti delle giornate di lavoro AIRO, pp. 1-17. 1989.
11. A. Linhares. "Industrial Pattern Sequencing Problems: Some Complexity Results And New Local Search Models". Doctoral Thesis, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, Brazil, 2002.
12. R. Möhring, "Graph problems related to gate matrix layout and PLA folding," Computing, Vol 7, pp. 17-51, 1990.
13. T. Kashiwabara, and T. Fujisawa, "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. 1979.
14. E. Fraggioli and C. A. Bentivoglio, "Heuristic and exact methods for the cutting sequencing problem.", European Journal of Operational Research, 110, pp. 564-575. 1998.
15. B J Yuen and K V Richardson "Establishing the optimality of sequencing heuristics for cutting stock problems". European Journal of Operational Research, 84, 590-598, 1995.