A biased random-key genetic algorithm for the Minimization of Open Stacks Problem *

José Fernando Gonçalves

LIAAD, INESC TEC, Faculdade de Economia do Porto, Universidade do Porto

Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal

jfgoncal@fep.up.pt

Mauricio G. C. Resende

AT&T Labs Research,

180 Park Avenue, Room C241, Florham Park, NJ 07932 USA

mgcr@research.att.com

Miguel Dias Costa

Faculdade de Economia do Porto, Universidade do Porto

Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal

migueldiascosta@gmail.com

This paper describes a biased random-key genetic algorithm (*BRKGA*) for the Minimization of Open Stacks Problem (*MOSP*). The *MOSP* arises in a production system scenario, and consists of determining a sequence of cutting patterns that minimizes the maximum number of opened stacks during the cutting process. The approach proposed combines a *BRKGA* and a local search procedure for generating the sequence of cut patterns. A novel fitness function for evaluating the quality of the solutions is also developed. Computational tests are presented using available instances taken from the literature. The high-quality of the solutions obtained validate the proposed approach.

Keywords: Minimization of Open Stacks Problem, Cutting Pattern, Biased Random-Key Genetic Algorithm; Random keys.

1 Introduction

Cutting stock problems consist in cutting smaller pieces (items) from larger pieces (objects) and arise in many industrial production scenarios, such as the furniture, paper, steel and wood hardboard industries. In the solution of cutting stock problems we seek to minimize waste or

* Ssupported by funds granted by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project PTDC/EGE-GES/117692/2010.
AT&T Labs Research Technical Report.
Date: 2013-06-27.

maximize profit through the selection of a set of good cutting patterns. However, in certain cases it is also important to determine the sequence in which the set of cutting patterns should be processed so as to minimize the maximum stack of partially cut orders. A cutting stock solution defines a set of cutting patterns and the number of times the patterns have to be cut to satisfy the demand for the items. When the patterns are cut, the items cut are piled up in stacks, one stack for each item type. The first time an item type is cut a stack is considered *open*. It remains open until the last piece of the corresponding item type is cut. A stack is *closed* when the last piece of an item type is cut. Because of space availability or of equipment limitations, which may force some stacks to be removed to free up space for the new stacks, it is desirable to maintain a small number of open stacks during the cutting process. Closed stacks can be moved to another location or can be delivered to clients. If removed, open stacks must be later returned so that work can be completed. This is inefficient since it takes time and uses scarce resources. To avoid the inefficiencies caused by the removal and later return of open stacks, it is important to determine the optimal cutting order of the patterns such that the maximum number of open stacks during the cutting process is minimized. This problem is known as the Minimization of Open Stacks Problem (*MOSP*).

To illustrate the *MOSP*, we use an example problem with five item types and four patterns. This problem is detailed in Table 1.

Table 1: Data for illustrative example *MOSP*.

Items	Patterns containing items
A	1, 3
B	1, 2, 4
C	1, 2
D	3
E	2, 4

Yanasse and Senne (2010) define *MOSP* as follows: Let M be a Boolean matrix where each row corresponds to an item type and each column corresponds to a cutting pattern. Each entry of $M_{i,k}$ (with $i = 1, \ldots, n$ and $k = 1, \ldots, m$) equals 1 if and only if at least one item of type i is contained in pattern$k$. Let M_s^1 be the resulting matrix corresponding to the permutation s of the columns of M such that in any row of M_s^1 each 0 entry between two 1 entries are replaced by a 1. Figure 1 depicts matrices M and M^1 corresponding to the permutation $s = (1, 2, 3, 4)$ for the example presented in Table 1.

a) - *M*

Items	Patterns 1	2	3	4
A	1	0	1	0
B	1	1	0	1
C	1	1	0	0
D	0	0	1	0
E	0	1	0	1

b) - *M¹*

Items	Patterns 1	2	3	4
A	1	1	1	0
B	1	1	1	1
C	1	1	0	0
D	0	0	1	0
E	0	1	1	1
Op. Stacks	3	4	4	2

Figure 1: M and M^1 corresponding to permutation $s = (1, 2, 3, 4)$ for the example of Table 1.

The objective of *MOSP* is to find a permutation s^* of the columns, such that the maximum number of 1 entries in any column of matrix $M^1_{s^*}$ is minimized. Figure 1 depicts the best solution for the example. It corresponds to the permutation $s = (3, 1, 2, 4)$ and has a *MOSP* value of three.

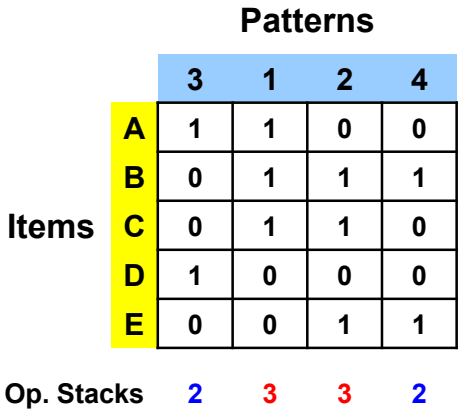

Figure 2: M^1 corresponding to the best permutation $s = (3, 1, 2, 4)$ for the example in Table 1.

Though the interest by the operations research community in the *MOSP* increased after the realization of the 2005 Constraint Modeling Challenge in the Fifth Workshop on Modelling and Solving Problems with Constraints, which focused on the *MOSP*, the number of publications on this problem is still not extensive.

The special case of *MOSP* where there are at most two different item types per pattern was considered by Lins (1989). While trying to solve a problem faced by the Australian glass industry Yuen (1991; 1995) developed six simple heuristics for the *MOSP*. The third heuristic (called XXX) was considered the most efficient in computational tests. Yuen and Richardson (1995) proposed the simple lower bound for the *MOSP* of the maximum number of different item types in the patterns and an exact method that enumerates permutations of pattern sequences. The new lower bound and the upper bounds provided by the heuristics of Yuen (1991; 1995) were used to reduce the search space for the exact method. Yanasse (1996) proposes polynomial-time algorithms for *MOSP* instances with special topologies. Yanasse (1997b), Limeira (1998) and Yanasse and Limeira (2004) propose branch and bound algorithms to solve the *MOSP*. Yanasse (1997a) defines the *MOSP* as a graph problem and shows that any *MOSP* instance corresponding to the same *MOSP* graph are equivalent. Faggioli and Bentivoglio (1998) develop a mathematical formulation for the *MOSP* and a solution method involving three phases. The first phase finds a good solution with a greedy heuristic similar to some of the heuristics of Yuen (1995). The second phase improves the solution obtained in the first phase using a tabu search. The third phase uses an implicit enumeration scheme of the permutations of patterns. Yanasse et al. (1999) and Becceneri (1999) propose arc contraction heuristics. A new lower bound for the optimal value of the *MOSP* is presented in Yanasse et al. (1999). Becceneri (1999) proposed the least-cost node heuristic for the *MOSP* which was later modified in Becceneri et al. (2004).

Metaheuristic-based heuristics have also been used to solve the MOSP. A simulated annealing heuristic is proposed in Linhares et al. (1999) and simulated annealing and tabu search are used in Fink and Voß (1999). A constructive genetic algorithm is proposed in Oliveira and Lorena (2002). Yanasse et al. (2007) proposed exact and heuristic methods using properties of the solution of *MOSP* to establish partial orders in which the nodes in the graph should be closed. An adaptive genetic algorithm for large-size open stack problems is presented in De Giovanni et al. (2010) and De Giovanni et al. (2013)

Dynamic-programming solutions to *MOSP* were developed by Banda and Stuckey (2007) and Chu and Stuckey (2009) where the search was simplified through the use of the properties presented in Becceneri et al. (2004) and Yuen and Richardson (1995).

Problems equivalent to the *MOSP* can arise in completely different contexts such as VLSI design (the gate matrix layout problem, the one dimensional logic, and PLA folding) and graph

theory (interval thickness, node search game, edge search game, graph path-width, narrowness, split bandwidth, edge separation, and vertex separation). See Linhares and Yanasse (2002) and Möhring (1990).

Yanasse and Senne (2010) presents a review of some properties and their use in pre-processing operations for the *MOSP*.

The MOSP is known to be NP-hard (Linhares and Yanasse, 2002). Therefore, when large instances are considered, heuristics are often the methods of choice. In this paper, we present a novel biased random-key genetic algorithm (BRKGA) for the MOSP. The proposed algorithm hybridizes a local search procedure with a genetic algorithm based on random keys. The BRKGA is used to evolve the order in which the patterns are inserted in a partial solution. To evaluate the quality of the solutions a novel fitness function is also developed.

The remainder of the paper is organized as follows. In Section 2 we introduce the new approach, describing in detail the BRKGA, the local search procedure, and the novel fitness function. Finally, in Section 3, we report on computational experiments, and in Section 4 make concluding remarks.

2 Biased random-key genetic algorithm

In this section we present an overview of the proposed solution process. This is followed by a discussion of the biased random-key genetic algorithm, including detailed descriptions of the solution encoding and decoding, evolutionary process, fitness function, and parallel implementation.

2.1 Overview

The new approach is based on a constructive heuristic algorithm which inserts patterns, one at a time, in a partial pattern sequence for the problem. Once all the patterns are inserted, a solution is obtained. The new approach proposed in this paper combines a biased random-key genetic algorithm, a local search procedure, and a novel fitness function. The role of the genetic algorithm is to evolve the encoded solutions, or *chromosomes*, which represent the *pattern insertion sequence* (*PIS*). For each chromosome, the following phases are applied to decode the chromosome:

1. *Decoding of the pattern insertion sequence*: This first phase decodes the chromosome into the *PIS*, i.e. the sequence in which the pattern are inserted into the partial pattern sequence.

2. *Construction of a solution*: The second phase makes use of the *PIS* defined in phase 1 and a local search procedure to construct a pattern sequence solution.

3. *Chromosome Adjustment*: The third phase adjusts the genes of the chromosome to reflect the changes made in phase 2.

4. *Fitness evaluation*: The final phase computes the fitness of the solution (or measure of quality of the solution). For this phase we developed a novel measure of fitness which significantly improves the quality of the solutions.

Figure 3 illustrates the sequence of decoding steps applied to each chromosome generated by the BRKGA.

The remainder of this section describes in detail the genetic algorithm, the decoding procedure, the local search, and the adjustment procedure.

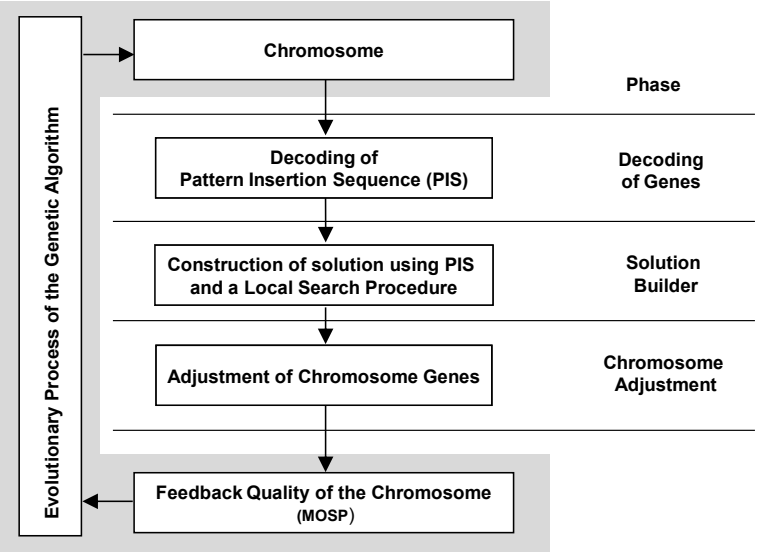

Figure 3: Architecture of the algorithm.

2.2 Biased random-key genetic algorithm

Random-key genetic algorithms (RKGA) or genetic algorithms with random keys were introduced in Bean (1994) for solving sequencing or optimization problems whose solutions can be represented as permutations. In a RKGA, chromosomes are represented as vectors of randomly generated real numbers in the interval $[0, 1]$. A deterministic algorithm, the *decoder*, takes as input a chromosome and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

Random key GAs are particularly attractive for sequencing problems and/or when the chromosomes have several parts (see for example Gonçalves and Almeida (2002), Gonçalves and Resende (2004), Gonçalves et al. (2005), GonÃ§alves et al. (2009), Goncalves and Sousa (2011), Gonçalves and Resende (2012), Gonçalves and Resende (2013), and Morán-Mirabal et al. (2013). Unlike traditional GAs, which need to use special repair procedures to handle permutations or sequences, RKGAs move all the feasibility issues into the objective evaluation procedure and guarantee that all offspring formed by crossover are feasible solutions. When the chromosomes have several parts, traditional GAs need to use different genetic operators for each part. However, since RKGAs use the *parametrized uniform crossover* of Spears and Dejong (1991)(instead of the traditional one-point or two-point crossover), they do not need to have different genetic operators for each part.

A *RKGA* evolves a *population* of random-key vectors over a number of *generations* (iterations). The initial population is made up of p vectors of r random keys. Each component of the solution vector, or random key, is generated independently at random in the real interval $[0, 1]$. After the fitness of each individual is computed by the decoder in generation g, the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e. those with the best fitness values, and the remaining set of $p - p_e$ *non-elite* individuals. To evolve a population g, a new generation of individuals is produced. All elite individual of the population of generation g are copied without modification to the population of generation $g + 1$. *RKGAs* implement mutation by introducing *mutants* into the population. A mutant is a vector of random keys generated in the same way that an element of the initial population is generated. At each generation, a small number p_m of mutants is introduced into the population. With $p_e + p_m$ individuals accounted for in population $g + 1$, $p - p_e - p_m$ additional individuals need to be generated to complete the p individuals that make up population $g + 1$. This is done by producing $p - p_e - p_m$ offspring solutions through the process of *mating* or *crossover*.

A *biased random-key genetic algorithm,* or *BRKGA* (Gonçalves and Resende, 2011), differs from a *RKGA* in the way parents are selected for mating. While in the *RKGA* of Bean (1994)

both parents are selected at random from the entire current population, in a *BRKGAs* each element is generated combining a parent selected at random from the elite partition in the current population and one is selected at random from the rest of the population. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring in the same generation. As in *RKGAs*, parameterized uniform crossover is used to implement mating in *BRKGAs*. Let ρ_e be the probability that the vector component of an elite parent is inherited by the offspring. For $i = 1, \ldots, r$, the i-th component $c(i)$ of the offspring vector c takes on the value of the i-th component $e(i)$ of the elite parent e with probability ρ_e and the value of the i-th component $\bar{e}(i)$ of the non-elite parent $\bar{e}$ with probability $1 - \rho_e$.

When the next population is complete, the corresponding fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation.

A *BRKGA* searches the solution space of the combinatorial optimization problem indirectly by searching the r-dimensional continuous hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated.

To specify a biased random-key genetic algorithm, we simply need to specify how solutions are encoded and decoded and how their corresponding fitness values are computed. We specify our algorithm next by first showing how the solutions of a *MOSP* are encoded and then decoded and how their fitness evaluation is computed.

2.2.1 Chromosome representation and decoding

A chromosome encodes a solution to the problem as a vector of random keys. In a direct representation, a chromosome represents a solution of the original problem, and is called *genotype*, while in an indirect representation it does not and special procedures are needed to obtain from it a solution called a *phenotype*. In the present context, the solutions will be represented indirectly by parameters that are later used by a decoding procedure to obtain a solution. To obtain the solution (phenotype) we use the decoding procedures described in Section 2.2.2.

In this paper, a solution to the *MOSP* is represented indirectly by the following chromosome structure:

$$chromosome = (gene_1, \ldots, gene_m)\,,$$

where m is the number of patterns. The decoding (mapping) of the m genes of each chromosome into a pattern insertion sequence (*PIS*), that will be used by the solution builder (see Section 2.2.2) is accomplished by sorting the patterns in ascending order of the corresponding gene values. Figure 4 shows an example of the decoding process for the *PIS*. In this example there are eight patterns. The sorted genes correspond to the $PIS = (5, 8, 3, 1, 4, 2, 6, 7)$.

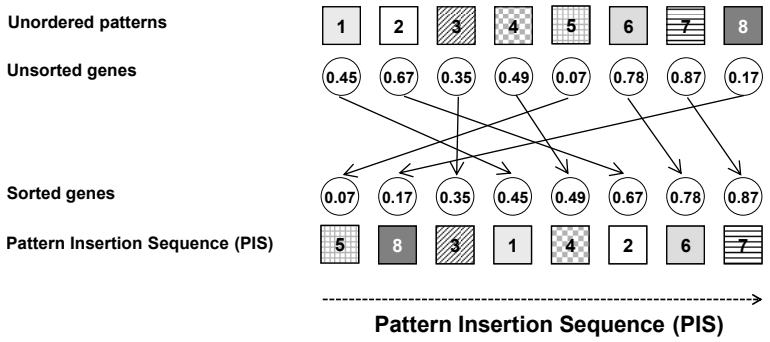

Figure 4: Decoding of the Pattern Insertion Sequence (*PIS*).

2.2.2 Solution builder

The solution builder follows a sequential process that inserts patterns into a partial solution, one pattern at each stage. The order in which the patterns are inserted into the partial solution is defined by the PIS evolved by the $BRKGA$. Each stage is comprised of the following two main steps:

1. Selection of pattern to be inserted;
2. Selection of the insertion position in the partial solution of the pattern selected in step 1) ;

The pattern selected for insertion at each stage j is given by PIS_j. The position in the partial solution where pattern PIS_j will be inserted is defined by a local search procedure. Let m_j be the number of patterns already in the partial solution at stage j. Then the local search procedure considers the insertion of pattern PIS_j before all existing patterns in positions $l = 1, \ldots, j-1$ and after the pattern in position $j-1$. The insertion position l_j^*, corresponding to the smallest value of $MOSP$ is selected as the insertion position. Pattern PIS_j is inserted at position l_j^* and the process is repeated until all the patterns are inserted.

2.2.3 Chromosome Adjustment

Solutions produced by the local search procedure usually disagree with the genes initially supplied to the decoder to obtain the PIS. Changes in the order of the patterns made by the local search phase of the decoder need to be taken into account in the chromosome. The heuristic adjusts the chromosome to reflect these changes. To make the chromosome supplied by the GA agree with the solution produced by local search, the heuristic adjusts the order of the genes according to the position of each pattern in the final solution. This chromosome adjustment not only improves the quality of the solutions but also reduces the number of generations needed to obtain the best values.

2.2.4 Fitness function

The evolutionary process requires a measure of solution fitness, or quality measure. A natural fitness function for the $MOSP$ is the *maximum number of open stacks* $MOSP$, in a solution. However, since different solutions can have the same $MOSP$ value, this measure does not differentiate well the potential for improvement of solutions having the same $MOSP$ value.

To better differentiate the potential for improvement we propose a new measure of fitness which we call *modified maximum number of open stacks*, or simply $MMOSP$. The $MMOSP$ combines $MOSP$ with a measure of the potential for improvement of a solution which has values in the interval $]0, 1]$. The rationale for this new measure is that if we have two solutions that have the same $MOSP$ value, then the one having the smallest average number of open stacks will have more potential for improvement.

Let $MOSP_K$ be the number of open stacks when pattern k is being cut. Let

$$\frac{1}{m}\sum_{k=1}^{m} MOSP_k$$

be the average number of open stacks. Then, the value of the modified maximum number of open stacks, $MMOSP$, is given by

$$MMOSP = MOSP + \frac{\sum_{k=1}^{m} MOSP_k}{m \times MOSP}.$$

The computational results in Section 3 show that this novel measure of fitness significantly improves the quality of the solutions. Figure 5a and 5b examplify the calculation of $MMOSP$ for two solutions having a $MOSP$ value equal to 4. The first solution (Figure 5a) has a $MMOSP$ equal to 4.8125 and the second solution (Figure 5b) has a $MMOSP$ value of 4.75.

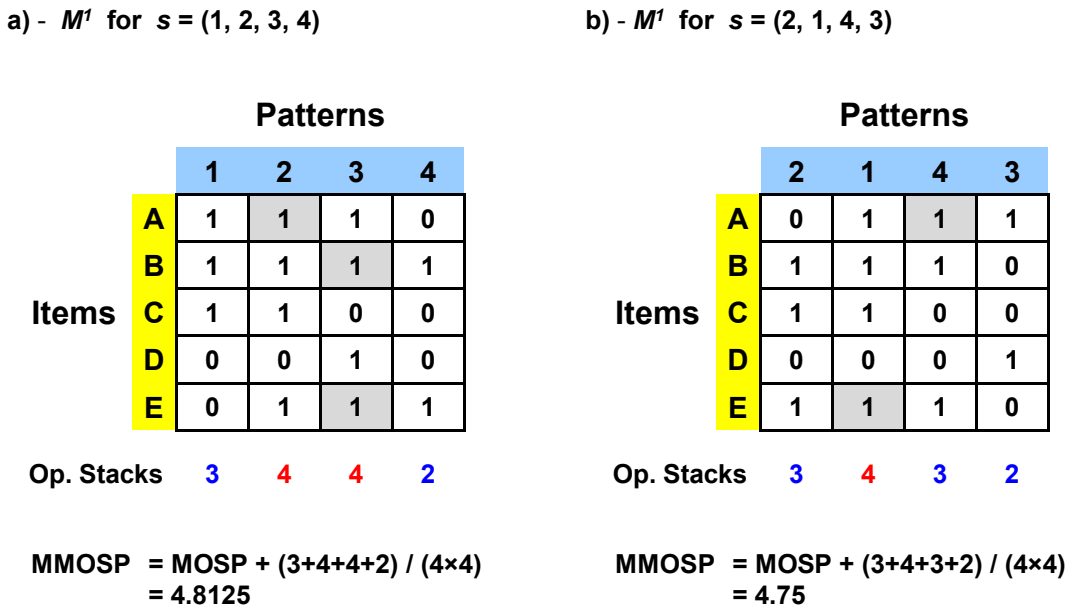

Figure 5: Example of the calculation of $MMOSP$.

3 Experimental results

In this section we report on results obtained on a set of experiments conducted to evaluate the performance of the biased random key genetic algorithm for MOSP (BRKGA-MOSP) proposed in this paper.

3.1 Benchmark algorithms

We compare *BRKGA-MOSP* with the approaches listed in Table 2. These are, to date, the most effective approaches found in the literature.

Table 2: Efficient approaches used for comparison.

Approach	Type of method	Source of approach
GHP	Greedy heuristic procedure	Faggioli and Bentivoglio (1998)
TS	Tabu search	Faggioli and Bentivoglio (1998)
GLS	Generalized local search	Faggioli and Bentivoglio (1998)
YUEN-3 and YUEN-5	Heuristics	Yuen (1995)
DP1	Dynamic programming	Banda and Stuckey (2007)
DP2	Dynamic programming	Chu and Stuckey (2009)
CGA	Constructive Genetic Algorithm,	Oliveira and Lorena (2002)
PMA	Parallel Memetic Algorithm	Mendes and Linhares (2004)
ECS	Evolutionary Clustering Search	Oliveira and Lorena (2006)
GRACS	Greedy Randomized Adaptive Clustering Search	Oliveira and Lorena (2006)
AGA	Adaptive Genetic Algorithm	De Giovanni et al. (2010) and De Giovanni et al. (2013)

3.2 Test problem instances

In the computational tests we used the instances described in Table 3

Table 3: Benchmark instances used in the computational tests.

Class	Description	Source
Harvey	2130 random instances by Harvey. Three benchmarks are proposed (denoted "wbo", "wbop", and "wbp"), each containing 10 classes ;	Smith and Gent (2005)
Simonis	3630 random instances by Simonis, grouped in 10 classes;	Smith and Gent (2005)
Shaw	one class of 25 random instances by Shaw, with 20 patterns and item types; .	Smith and Gent (2005)
Miller and Wilson	21 individual instances, one provided by Miller and 20 by Wilson (denoted GP 1–8, NWRS 1–8, SP 1–4);	Smith and Gent (2005)
Faggioli and Bentivoglio	300 random instances with $n = 10, 20, 30, 40, 50$ and $m = 10, 15, 20, 25, 30, 40$. Each of the $n \times m$ combinations was replicated 10 times.	Faggioli and Bentivoglio (1998)
VLSI	11 individual instances from the VLSI industry ;	Hu and Chen (1990)
SCOOP	24 real instances from two woodcutting companies (denoted "A" and "B")	available from http://www.scoop-project.net.

3.3 GA configuration

The configuration of genetic algorithms is oftentimes more an art form than a science. In our past experience with genetic algorithms based on the same evolutionary strategy (see Gonçalves et al. (2005), GonÃ§alves et al. (2009), GonÃ§alves et al. (2011), Gonçalves and Resende (2012), and Gonçalves and Resende (2013)), we obtained good results with values of TOP, BOT, and $CrossoverProbability$ ($CProb$) in the intervals shown in Table 4.

Table 4: Range of Parameters in past implementations.

Parameter	Interval
TOP	0.10 - 0.25
BOT	0.15 - 0.30
Crossover Probability (CProb)	0.70 - 0.80

For the population size, we obtained good results by indexing it to the dimension of the problem, i.e. we used small size populations for small problems and larger populations for larger problems. The configuration presented in Table 5 was held constant for all experiments and all problem instances. The computational results presented in the next section demonstrate that this configuration not only provides excellent results in terms of solution quality but is also very robust.

Table 5: Configuration parameters for the $BRKGA - MOSP$ algorithm.

Parameter	Value
$p =$	$10 \times m$
$p_e =$	$min(0.25 \times p,50)$
$p_m =$	$0.20 \times p$
$\rho_e =$	0.70
Fitness =	$MMOSP$ = Modified MOSP (to minimize)
Stopping Criterion =	100 generations

3.4 Computational results

Algorithm *BRKGA* was implemented in C++ and the experiments were carried out on a computer with a Intel Core i7-2630QM @2.0 GHZ CPU running the Linux operating system with Fedora release 16.

Before running the BRKGA we applied a pre-processing step to each instance as described in Becceneri et al. (2004).

Due to the non-deterministic nature of BRKGA, 10 runs have been considered for each instance, and best results are used for comparison. Tables 6-11 present the results obtained for the various instance classes (6141 instances) by the BRKGA and some of the other approaches. In the tables each row is associated with a class of aggregated instances or an individual instance. The first columns define instance name and size, optimal values, etc. The columns denoted by MOSP represent the best MOSP found by the corresponding approach. In terms of computational times, we cannot make any fair and meaningful comment since all the other approaches were implemented with different programming languages and tested on computers with different computing power. Hence, to avoid discussion about the different computers speed used in the tests, we limit ourselves to reporting the average running times per run for BRKGA, while for each of the other algorithms we only report, when available, the reported running times. When available the columns with header $T(s)$ represent the running time of the corresponding approaches, in seconds.

BRKGA dominates all other approaches with respect to the quality of the solution and is very competitive in terms of running time. MOSP always finds the optimal or best known solution of MOSP in reasonable running time (often negligible), while the computational effort as well as the quality of the solutions of DP1 degrade with large size instances SP3 and SP4. DP2 overcomes this issue and solves instances SP2, SP3, and SP4 very quickly. For most of the instances, the BRKGA obtains the best solution before the tenth generation.

Table 6: Computational results: Harvey, Simonis and Shaw benchmarks (aggregate results).

	I	P		BRKGA		AGA		DP1	
	n	m	Elements	MOSP	T(s)	MOSP	T(s)	MOSP	T(s)
Simonis	10	10	550	8.0	0.00	8.0	0.00	8.0	0.00
	10	20	550	8.9	0.00	8.9	0.01	8.9	0.00
	15	15	550	12.9	0.00	12.9	0.02	12.9	0.00
	15	30	550	14.0	0.00	14.0	0.10	14.0	0.00
	20	10	220	15.9	0.00	15.9	0.03	15.9	0.00
	20	20	550	18.0	0.00	18.0	0.11	18.0	0.01
	30	10	220	24.0	0.00	24.0	0.05	24.0	0.00
	30	15	110	26.0	0.00	26.0	0.11	26.0	0.01
	30	30	220	28.3	1.30	28.3	0.61	28.3	0.72
	40	20	110	36.4	0.43	36.4	0.33	36.4	0.10
Shaw	20	20	25	13.7	0.00	13.7	0.34	13.7	0.01
wbo	10	10	40	5.9	0.00	5.9	0.00	5.9	0.00
	10	20	40	7.4	0.00	7.4	0.02	7.4	0.00
	10	30	40	8.2	0.00	8.2	0.08	8.2	0.00
	15	15	60	9.4	0.00	9.4	0.06	9.4	0.00
	15	30	60	11.6	0.00	11.6	0.39	11.6	0.03
	20	10	70	12.9	0.00	12.9	0.04	12.9	0.00
	20	20	90	13.7	0.00	13.7	0.22	13.7	0.01
	30	10	100	20.1	0.00	20.1	0.08	20.1	0.00
	30	15	120	21.0	0.00	21	0.18	21.0	0.01
	30	30	140	22.6	1.08	22.6	1.11	22.6	1.11
wbop	10	10	40	6.8	0.00	6.8	0.00	6.8	0.00
	10	20	40	8.1	0.00	8.1	0.04	8.1	0.00
	10	30	40	8.6	0.00	8.6	0.06	8.6	0.00
	15	15	60	10.4	0.00	10.4	0.05	10.4	0.00
	15	30	60	12.2	0.00	12.2	0.34	12.2	0.02
	20	10	40	14.3	0.00	14.3	0.02	14.3	0.00
	20	20	90	14.9	0.00	14.9	0.15	14.9	0.01
	30	10	40	22.5	0.00	22.5	0.05	22.5	0.00
	30	15	60	22.4	0.00	22.4	0.13	22.4	0.01
	30	30	140	23.8	0.00	23.9	0.95	23.8	0.99
wbp	10	10	40	7.3	0.00	7.3	0.00	7.3	0.00
	10	20	70	8.7	0.00	8.7	0.02	8.7	0.00
	10	30	100	9.3	0.00	9.3	0.03	9.3	0.00
	15	15	60	11.1	0.00	11.1	0.04	11.1	0.00
	15	30	120	13.1	0.00	13.1	0.18	13.1	0.01
	20	10	40	15.1	0.00	15.1	0.03	15.1	0.00
	20	20	90	15.4	0.00	15.4	0.13	15.4	0.01
	30	10	40	23.2	0.08	23.2	0.06	23.2	0.00
	30	15	60	23.0	0.19	23.0	0.14	23.0	0.01
	30	30	140	24.5	0.98	24.5	0.74	24.5	1.20

Table 7: Computational results: Miller and Wilson benchmarks (individual instances).

			BRKGA		DP1		DP2		AGA	
Instance	I	P	MOSP	T(s)	MOSP	T(s)	MOSP	T(s)	MOSP	T(s)
Miller	20	40	13	0.0	13	0.6	-	-	13	2.68
GP1	50	50	45	0.0	45	0.0	-	-	45	0.02
GP2	50	50	40	0.0	40	0.0	-	-	40	0.04
GP3	50	50	40	0.0	40	0.0	-	-	40	0.24
GP4	50	50	30	0.0	30	0.0	-	-	30	0.02
GP5	100	100	95	0.5	95	0.1	-	-	95	0.44
GP6	100	100	75	0.7	75	0.1	-	-	75	0.62
GP7	100	100	75	0.7	75	0.1	-	-	75	0.58
GP8	100	100	60	0.7	60	0.2	-	-	60	0.58
NWRS1	10	20	3	0.0	3	0.0	-	-	3	0
NWRS2	10	20	4	0.0	4	0.0	-	-	4	0
NWRS3	15	25	7	0.0	7	0.0	-	-	7	0
NWRS4	15	25	7	0.0	7	0.0	-	-	7	0
NWRS5	20	30	12	0.0	12	0.0	-	-	12	0
NWRS6	20	30	12	0.0	12	0.0	-	-	12	0
NWRS7	25	60	10	0.0	10	0.0	-	-	10	0
NWRS8	25	60	16	0.0	16	2.1	-	-	16	6
SP1	25	25	9	0.0	9	0.0	-	-	9	0.45
SP2	50	50	19	0.0	19	1650.0	19	0.0	19	10.9
SP3	75	75	34	0.2	36	~3600	34	0.4	34	36.7
SP4	100	100	53	0.6	56	~14400	53	0.9	53	81.0

Table 8: Computational results: Faggioli and Bentivoglio's (1998) instances.

I	P							
n	m	OPT	BRKGA	GHP	TS	GLS	YUEN3	YUEN5
10	10	5.5	5.5	5.5	5.5	5.5	5.8	5.7
	15	6.6	6.6	6.6	6.6	6.6	7.0	7.0
	20	7.5	7.5	7.7	7.7	7.5	7.5	7.8
	25	8.0	8.0	8.0	8.0	8.0	8.2	8.0
	30	7.8	7.8	7.8	7.8	7.8	8.2	7.9
	40	8.4	8.4	8.4	8.4	8.4	8.6	8.4
20	10	6.2	6.2	6.6	6.2	6.2	6.8	6.7
	15	7.2	7.2	7.5	7.2	7.5	8.4	8.3
	20	8.5	8.5	8.8	8.7	8.6	10.1	9.5
	25	9.8	9.8	9.9	9.8	9.9	11.4	10.9
	30	11.1	11.1	11.4	11.2	11.2	12.7	12.1
	40	13,0	13.0	13.1	13.1	13.1	14.8	13.7
30	10	6.1	6.1	6.4	6.1	6.2	7.0	7.0
	15	7.4	7.3	8.0	7.4	7.6	9.1	8.6
	20	8.8	8.8	9.8	9.2	8.9	10.8	10.2
	25	10.5	10.5	11.1	10.7	10.6	12.8	12.2
	30	12,2	12.2	13.0	12.6	12.2	14.7	13.6
	40	14,5	14.5	15.0	14.7	14.6	17.3	15.9
40	10	7.7	7.7	7.9	7.7	7.7	8.0	7.9
	15	7.3	7.2	8.2	7.3	7.4	8.4	8.1
	20	8.5	8.5	9.5	8.6	8.7	10.4	9.9
	25	10.3	10.3	11.6	10.7	10.6	13.1	11.6
	30	12.1	12.1	13.4	12.6	12.4	15.1	14.6
	40	15.0	14.9	15.8	15.3	15.3	18.5	16.9
50	10	8.2	8.2	8.4	8.2	8.2	8.5	8.4
	15	7.4	7.4	8.4	7.6	7.6	8.4	8.1
	20	7.9	7.9	9.2	8.0	8.2	9.7	9.2
	25	10.0	10.0	11.2	10.1	10.2	12.3	12.0
	30	11.2	11.2	12.4	12.0	11.8	14.9	13.5
	40	14.6	14.6	16.8	15.3	14.9	18.5	17.5

Table 9: Computational results: VLSI instances.

	I	P		BRKGA		AGA		PMA		CGA		ECS		GRACS	
	n	m	BKS	MOSP	T(s)	MOSP	T(s)	MOSP	T(s)	MOSP	T(s)	MOSP	T(s)	MOSP	T(s)
v4470	37	47	9	9	2.8	9	5.3	9	10	9	66.5	9	-	9	-
x0	40	48	11	11	2.3	11	6.6	11	30	11	75.6	11	-	11	-
W2	48	33	14	14	0.7	14	0	14	30	14	18.5	14	-	14	-
W3	84	70	18	18	8.4	18	18.9	18	90	18	306.3	18	-	18	-
W4	202	141	27	27	47.3	27	67,2	27	2400	27	5224.7	27	-	27	-

Table 10: Computational results: real instances from company "A".

	I	P					
	n	m	OPT	BRKGA	T(s)	AGA	T(s)
A_AP-9.d_10	20	13	6	6	0.05	6	0.12
A_AP-9.d_3	20	16	6	6	0.05	6	0.17
A_FA+AA-_15	68	18	9	9	0.23	9	0.63
A_FA+AA-_2	75	19	11	11	0.15	11	0.68
A_AP-9.d_6	31	20	5	5	0.15	5	0.40
A_FA+AA-_12	75	20	9	9	0.23	9	0.77
A_AP-9.d_11	27	21	6	6	0.15	6	0.42
A_FA+AA-_6	79	21	13	13	0.28	13	1.20
A_FA+AA-_8	82	28	11	11	0.60	12	2.65
A_FA+AA-_11	99	28	11	11	0.65	11	2.92
A_FA+AA-_1	107	37	12	12	1.38	12	7.58
A_FA+AA-_13	134	37	-	17	1.45	17	11.82

Table 11: Computational results: real instances from company "B".

	I	P					
	n	m	OPT	BRKGA	T(s)	AGA	T(s)
B_39Q18_82	14	10	5	5	0.00	5	0.00
B_ 42F22_93	18	10	5	5	0.05	5	0.04
B_22X18_50	14	11	10	10	0.03	10	0.05
B_18AB1_32	15	11	6	6	0.03	6	0.05
B_CARLET_137	14	12	5	5	0.03	5	0.00
B_12F18_11	21	15	6	6	0.08	6	0.16
B_18CR1_33	20	18	4	4	0.03	4	0.17
B_GTM18A_139	24	20	5	5	0.10	5	0.30
B_ 23B25_52	29	21	5	5	0.10	5	0.39
B_ 12M18_12	31	22	6	6	0.15	6	0.00
B_ CUC28A_138	37	26	6	6	0.10	6	0.00
B_ REVAL_145	60	49	7	7	1.28	7	9.37

4 Concluding remarks

In this paper we addressed the Minimization of Open Stacks Problem which consists of determining a sequence of cutting patterns that minimizes the maximum number of opened stacks during the cutting process. The approach proposed combines a *BRKGA* and a local search procedure for generating the sequence of cutting patterns. A new fitness function for evaluating the quality of the solutions is also developed. Computational tests are presented using 6141 available instances taken from the literature. The high-quality of the solutions obtained validate the proposed approach.

The new approach is extensively tested on 6141 problem instances and compared with other approaches published in the literature. The computational experiments results demonstrate that the new approach consistently equals or outperforms the other approaches.

Acknowledgments

This work has been supported by funds granted by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project PTDC/EGE-GES/117692/2010.

References

Banda, M.G., Stuckey, P.J., 2007. Dynamic programming to minimize the maximum number of open stacks. INFORMS Journal on Computing 19, 607–617.

Bean, J.C., 1994. Genetics and random keys for sequencing and optimization. ORSA Journal on Computing 6, 154–160.

Becceneri, J., 1999. O problema de sequenciamento de padrões para minimização do número máximo de pilhas abertas em ambientes de corte industriais. Ph.D. thesis. Engenharia Eletrônica e Computação, ITA/CTA, São José dos Campos.

Becceneri, J.C., Yanasse, H.H., Soma, N.Y., 2004. A method for solving the minimization of the maximum number of open stacks problem within a cutting process. Computers & Operations Research 31, 2315–2332.

Chu, G., Stuckey, P.J., 2009. Minimizing the maximum number of open stacks by customer search, in: Principles and Practice of Constraint Programming-CP 2009. Springer. volume 5732 of *Lecture Notes in Computer Science*, pp. 242–257.

De Giovanni, L., Massi, G., Pezzella, F., 2010. Preliminary computational experiments with a genetic algorithm for the open stacks problem. Technical Report. Dipartimento di Matematica Pura ed Applicata Università degli studi di Padova, via Trieste, 63 - 35121 Padova (Italy).

De Giovanni, L., Massi, G., Pezzella, F., 2013. An adaptive genetic algorithm for large-size open stack problems. International Journal of Production Research 51, 682–697.

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

Fink, A., Voß, S., 1999. Applications of modern heuristic search methods to pattern sequencing problems. Computers and Operations Research 26, 17–34.

Gonçalves, J., Resende, M., 2012. A parallel multi-population biased random-key genetic algorithm for a container loading problem. Computers & Operations Research 39, 179–190.

Gonçalves, J.F., Resende, M.G., 2013. A biased random key genetic algorithm for 2d and 3d bin packing problems. International Journal of Production Economics .

Goncalves, J.F., Sousa, P.S.A., 2011. A genetic algorithm for lot sizing and scheduling under capacity constraints and allowing backorders. International Journal of Production Research 49, 2683–2703.

Gonçalves, J.F., Almeida, J.R., 2002. A hybrid genetic algorithm for assembly line balancing. Journal of Heuristics 8, 629–642.

Gonçalves, J.F., Mendes, J.J.M., Resende, M.G.C., 2005. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167, 77–95.

Gonçalves, J.F., Resende, M.G.C., 2004. An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering 47, 247–273.

Gonçalves, J.F., Resende, M.G.C., 2011. Biased random-key genetic algorithms forÂ combinatorial optimization. Journal of Heuristics 17, 487–525.

Gonçalves, J.F., Resende, M.G.C., 2013. An extended Akers graphical with a biased random-key genetic algorithm for job-shop scheduling. International Transactions in Operational Research To appear.

GonÃ§alves, J.F., Mendes, J.J.M., Resende, M.G.C., 2009. A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research 189, 1171–1190.

GonÃ§alves, J.F., Resende, M.G.C., Mendes, J.J.M., 2011. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics 17, 467–486.

Hu, Y.H., Chen, S.J., 1990. Gm plan: a gate matrix layout algorithm based on artificial intelligence planning techniques. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 9, 836–845.

Limeira, M., 1998. Desenvolvimento de um algoritmo exato para a solução de um problema de seqüenciamento de padrões de corte. 1998. Ph.D. thesis. Dissertação (Mestrado em Computação Aplicada), Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1998.[Links].

Linhares, A., Yanasse, H.H., 2002. Connections between cutting-pattern sequencing, vlsi design, and flexible machines. Computers & Operations Research 29, 1759–1772.

Linhares, A., Yanasse, H.H., Torreao, J.R., 1999. Linear gate assignment: a fast statistical mechanics approach. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 18, 1750–1758.

Lins, S., 1989. Traversing trees and scheduling tasks for duplex corrugator machines. Pesquisa Operacional 9, 40–54.

Mendes, A., Linhares, A., 2004. A multiple-population evolutionary approach to gate matrix layout. International Journal of Systems Science 35, 13–23.

Möhring, R.H., 1990. Graph problems related to gate matrix layout and PLA folding. Springer.

Morán-Mirabal, L., González-Velarde, J., Resende, M., 2013. Randomized heuristics for the family traveling salesperson problem. International Transactions in Operational Research .

Oliveira, A.C., Lorena, L.A., 2006. Pattern sequencing problems by clustering search, in: Advances in Artificial Intelligence-IBERAMIA-SBIA 2006. Springer, pp. 218–227.

Oliveira, A.C.M., Lorena, L.A.N., 2002. A constructive genetic algorithm for gate matrix layout problems. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 21, 969–974.

Smith, B., Gent, I., 2005. Constraint modelling challenge 2005, in: IJCAI 2005 Fifth Workshop on Modelling and Solving Problems with Constraints, pp. 1–8.

Spears, W.M., Dejong, K.A., 1991. On the virtues of parameterized uniform crossover, in: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236.

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

Yanasse, H., 1997a. A transformation for solving a pattern sequencing problem in the wood cut industry. Pesquisa Operacional 17, 57–70.

Yanasse, H., Becceneri, J., Soma, N., 1999. Bounds for a problem of sequencing patterns. Pesquisa Operacional 19, 249–277.

Yanasse, H., Limeira, M., 2004. Refinements on an enumeration scheme for solving a pattern sequencing problem. International Transactions in Operational Research 11, 277–292.

Yanasse, H.H., 1997b. On a pattern sequencing problem to minimize the maximum number of open stacks. European Journal of Operational Research 100, 454–463.

Yanasse, H.H., Becceneri, J.C., Soma, N.Y., 2007. Um algoritmo exato com ordenamento parcial para solução de um problema de programação da produção: experimentos computacionais. Gestão & Produção 14, 353–361.

Yanasse, H.H., Senne, E.L.F., 2010. The minimization of open stacks problem: A review of some properties and their use in pre-processing operations. European Journal of Operational Research 203, 559–567.

Yuen, B.J., 1991. Heuristics for sequencing cutting patterns. European Journal of Operational Research 55, 183–190.

Yuen, B.J., 1995. Improved heuristics for sequencing cutting patterns. European Journal of Operational Research 87, 57–64.

Yuen, B.J., Richardson, K.V., 1995. Establishing the optimality of sequencing heuristics for cutting stock problems. European Journal of Operational Research 84, 590–598.