INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 00 (2014) 1–22
DOI: 10.1111/itor.12109

A biased random-key genetic algorithm for the minimization of open stacks problem

José Fernando Gonçalves[a], Mauricio G. C. Resende[b] and Miguel Dias Costa[c]

[a]*LIAAD, INESC TEC, Faculdade de Economia do Porto, Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal*
[b]*AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932, USA*
[c]*Faculdade de Economia do Porto, Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal*
E-mail: jfgoncal@fep.up.pt [Gonçalves]; mgcr@research.att.com [Resende]; migueldiascosta@gmail.com [Costa]

Received 27 July 2013; received in revised form 27 May 2014; accepted 27 May 2014

Abstract

This paper describes a biased random-key genetic algorithm (*BRKGA*) for the minimization of the open stacks problem (*MOSP*). The *MOSP* arises in a production system scenario, and consists of determining a sequence of cutting patterns that minimize the maximum number of open stacks during the cutting process. The proposed approach combines a *BRKGA* and a local search procedure for generating the sequence of cutting 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.

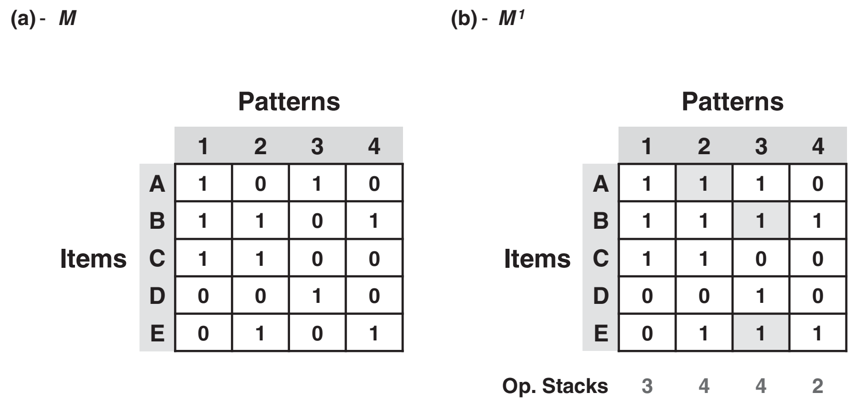
Keywords: minimization of open stacks problem; cutting pattern; biased random-key genetic algorithm; random keys

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 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 equipment limitations, which may force some stacks to be removed to free

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


(a) - M

Patterns

Items	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) - M^1

Patterns

Items	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

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

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 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.

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 $i = 1, \dots, n$ and $k = 1, \dots, m$) 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.

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 $M_{s^*}^1$ is minimized. Figure 2 depicts one optimal solution for the example. It corresponds to the permutation $s = (3, 1, 2, 4)$ and has an *MOSP* value of three.

Though the interest of the operations research community in the *MOSP* increased after the realization of the 2005 Constraint Modeling Challenge in the Fifth Workshop on Modeling and

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

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

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 Heuristic 3) 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 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 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 (GA) 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 GA for large-sized open stack problems is presented in De Giovanni et al. (2010, 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, 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; Möhring, 1990). Yanasse and Senne (2010) present a review of some properties and their use in preprocessing 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 GA (BRKGA) for the MOSP. The proposed algorithm hybridizes a local search procedure with a GA 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, local search procedure, and novel fitness function. Finally, in Section 3, we report computational experiments, and in Section 4 we make concluding remarks.

2. BRKGA

In this section, we present an overview of the proposed solution process. This is followed by a discussion of the BRKGA, 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 that 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 BRKGA, local search procedure, and novel fitness function. The role of the GA 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 PIS*: This first phase decodes the chromosome into the *PIS*, that is, the sequence in which the patterns 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 that 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 GA, decoding procedure, local search, and adjustment procedure.

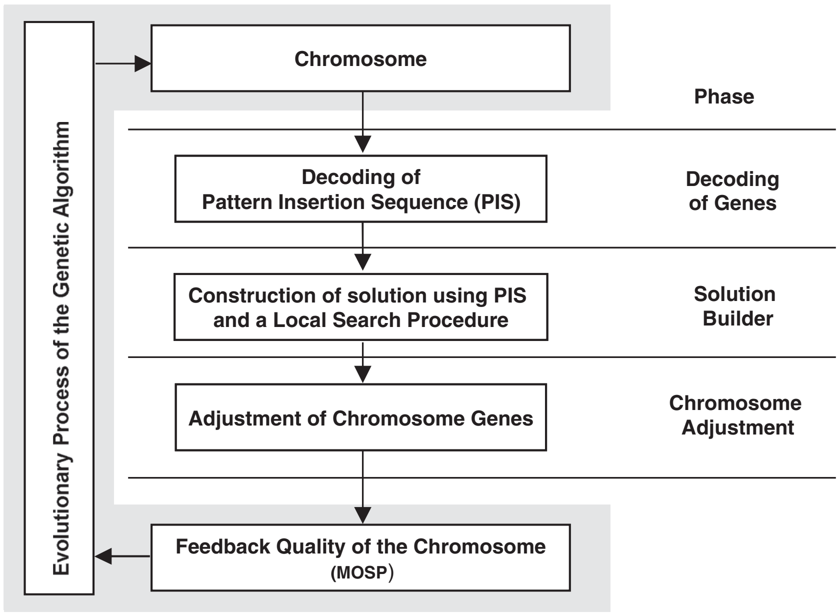

Evolutionary Process of the Genetic Algorithm
Chromosome
Phase
Decoding of Pattern Insertion Sequence (PIS)
Decoding of Genes
Construction of solution using PIS and a Local Search Procedure
Solution Builder
Adjustment of Chromosome Genes
Chromosome Adjustment
Feedback Quality of the Chromosome (MOSP)

Fig. 3. Architecture of the algorithm.

2.2. BRKGA

"Random-key GAs" (RKGAs) or GAs with random keys were introduced in Bean (1994) for solving sequencing or optimization problems whose solutions can be represented as permutations. In an 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.

RKGAs are particularly attractive for sequencing problems and/or when the chromosomes have several parts (see, e.g. Gonçalves and Almeida, 2002; Gonçalves and Resende, 2004, 2012, 2013; Gonçalves et al., 2005, 2009; Gonçalves and Sousa, 2011; Morán-Mirabal et al., 2014). Unlike the traditional GAs that use special repair procedures to handle permutations or sequences, RKGAs move all the feasibility issues to 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.

An *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 divided into two groups of individuals: a small group of p_e "elite" individuals, that is, those with the best fitness values, and the remaining set of $p - p_e$ "nonelite" individuals. To evolve a population g, a new generation of individuals is produced. All elite individuals of the population of generation g are copied without modification to the population of generation $g + 1$. *RKGA*s implement mutation by introducing "mutants" into the population. A mutant is a vector of random keys generated in the same way in which 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 "BRKGA" (Gonçalves and Resende, 2011) differs from an *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 *BRKGA*s 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 *RKGA*s, parameterized uniform crossover is used to implement mating in *BRKGA*s. 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 ith component $c(i)$ of the offspring vector c takes on the value of the ith component $e(i)$ of the elite parent e with probability ρ_e and the value of the ith component $\bar{e}(i)$ of the nonelite parent $\bar{e}$ with probability $1 - \rho_e$.

When the next population is complete, the corresponding fitness values are computed for all the newly created random-key vectors and the population is divided into elite and nonelite 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 BRKGA, we simply need to specify how solutions are encoded and decoded, and how their corresponding fitness values are computed. Next we specify our algorithm by first showing how the solutions of an *MOSP* are encoded and then decoded, and how their fitness evaluation is computed.

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 "Solution builder."

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

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

where m is the number of patterns. The decoding (mapping) of the m genes of each chromosome into a *PIS*, which will be used by the solution builder (see Section "Solution builder"), is accomplished

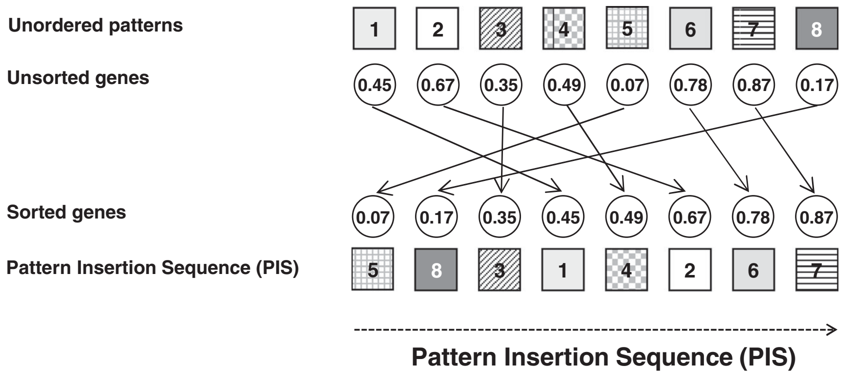


Fig. 4. Decoding of the pattern insertion sequence (*PIS*).

by sorting the patterns in an 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 $PIS = (5, 8, 3, 1, 4, 2, 6, 7)$.

Solution builder

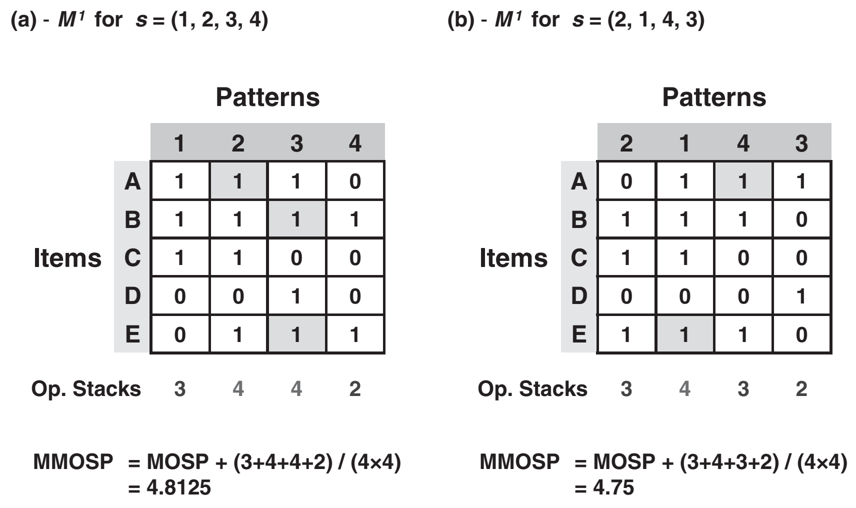
The solution builder follows a sequential process that inserts patterns into a partial solution, one pattern at each stage. The order in which the patterns are inserted into the partial solution is defined by the PIS evolved by the $BRKGA$. Each stage comprises the following two main steps:

(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.

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.



(a) - M^1 for $s = (1, 2, 3, 4)$

Items	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

MMOSP = MOSP + (3+4+4+2) / (4×4)
= 4.8125

(b) - M^1 for $s = (2, 1, 4, 3)$

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

MMOSP = MOSP + (3+4+3+2) / (4×4)
= 4.75

Patterns
Patterns

Fig. 5. Example of the calculation of *MMOSP*.

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 that 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 that 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 *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 b exemplifies the calculation of *MMOSP* for two solutions

Table 2
Other 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)
AS	Low-order polynomial time heuristic	Ashikaga and Soma (2009)
AGA	Adaptive genetic algorithm	De Giovanni et al. (2010, 2013)

having an $MOSP$ value equal to 4. The first solution (Fig. 5a) has an $MMOSP$ equal to 4.8125 and the second solution (Fig. 5b) has an $MMOSP$ value of 4.75.

3. Experimental results

In this section, we report the results obtained on a set of experiments conducted to evaluate the performance of the BRKGA for MOSP proposed in this paper.

3.1. Benchmark algorithms

We compare *BRKGA* with the literature approaches listed in Table 2.

3.2. Test problem instances

In the computational tests, we used three sets of instances described in Table 3.

3.3. GA configuration

The configuration of GAs is oftentimes more an art form than a science. In our past experience with GAs based on the same evolutionary strategy (see Gonçalves et al., 2005, 2009, 2011; Gonçalves and Resende, 2012, 2014), we obtained good results with values of TOP, BOT, and "crossover probability" ($CProb$) in the intervals shown in Table 4.

Table 3
Benchmark instances used in the computational tests

Set	Class	Description	Source
Literature instances	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
Harder instances	Harder150	150 harder instances generated using the procedure detailed in (Chu and Stuckey, 2009)	De Giovanni et al. (2013)
Becceneri instances	Becceneri	Set of instances that covers the most common practical industrial scenario applications	Becceneri (1999)

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

Table 5
Configuration parameters for the *BRKGA* algorithm

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

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

	I	P		BRKGA		AGA		DP1	
	n	m	Elements	MOSP	T (seconds)	MOSP	T (seconds)	MOSP	T (seconds)
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 (seconds)	MOSP	T (seconds)	MOSP	T (seconds)	MOSP	T (seconds)
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	9.1	53	81.0

For the population size, we obtained good results by indexing it to the dimension of the problem, that is, we used small-sized 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.

3.4. Computational results

Algorithm *BRKGA* was implemented in C++ and the experiments were carried out on a computer with Intel Core i7-2630QM @2.0 GHZ CPU running the Linux operating system with Fedora release 16. Before running the *BRKGA*, we applied a preprocessing step to each instance as described in Becceneri et al. (2004).

Literature instances

Due to the nondeterministic nature of BRKGA, 10 runs have been considered for each instance, and the best results are used for comparison. Tables 6–11 present the results obtained by the BRKGA and some of the other approaches for the various instance classes (6141 instances) included in the

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

I / *n*	*P* / *m*	OPT	BRKGA	GHP	TS	GLS	YUEN3	YUEN5
10	10	5.5	5.5	5.5	513.5	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

set "Literature instances." 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 computer 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 (seconds) 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. BRKGA always finds the optimal or best-known solution of

Table 9
Computational results: VLSI instances

	I	*P*		BRKGA		AGA		PMA		CGA		ECS		GRACS	
	n	*m*	BKS	MOSP	*T* (seconds)	MOSP	*T* (seconds)	MOSP	*T* (seconds)	MOSP	*T* (seconds)	MOSP	*T* (seconds)	MOSP	*T* (seconds)
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"

	$\frac{I}{n}$	$\frac{P}{m}$	OPT	BRKGA	T (seconds)	AGA	T (seconds)
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"

	$\frac{I}{n}$	$\frac{P}{m}$	OPT	BRKGA	T (seconds)	AGA	T (seconds)
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

MOSP in reasonable running time (often negligible), while the computational effort as well as the quality of the solutions of DP1 degrade with large-sized 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 10th generation.

Harder instances

In this section, we use a total of 150 harder instances to evaluate the performance of BRKGA. These instances were generated by De Giovanni et al. (2013) using the same random generator as Chu and Stuckey (2009). For each triplet defined by the number of patterns chosen in the set {50, 100, 150},

Table 12
Computational results: harder instances

I	*P*	*D*	AGA time	*BRKGA*						
n	*m*		(seconds)	/1	/2	/3	/4	AGA	PMA	DP2
		2	33.8	**13.2**	**13.2**	**13.2**	**13.2**	13.4	14.2	13.8
		4	42.7	**29.6**	**29.6**	**29.6**	**29.6**	30.4	29.8	29.8
200	50	6	53.2	**48.0**	**48.0**	**48.0**	**48.0**	49.0	48.2	49.0
		8	56.1	62.8	62.8	62.8	62.8	63.0	**62.4**	63.4
		10	56.8	**78.6**	**78.6**	**78.6**	**78.6**	79.0	79.0	80.4
		2	106.7	**19.4**	**19.6**	**19.6**	**19.6**	20.8	22.4	19.4
		4	140.6	**49.4**	**49.6**	**49.6**	**50.0**	51.6	51.4	51.2
200	100	6	164.8	**82.8**	**82.8**	**82.8**	**83.0**	85.6	85.6	84.8
		8	200.3	**109.0**	**109.0**	**109.0**	**109.0**	110.0	110.0	110.2
		10	216.1	**132.0**	**132.0**	**132.0**	**132.0**	135.0	132.2	132.2
		2	159.6	**26.8**	**26.8**	**27.0**	**27.0**	29.4	30.8	**27.0**
		4	200.3	**67.4**	**67.6**	**67.8**	**67.8**	71.4	72.2	69.0
200	150	6	246.0	**107.0**	**107.0**	**107.0**	**107.0**	110.2	108.4	108.0
		8	278.3	**133.6**	**133.6**	**133.6**	**133.6**	138.4	137.2	134.0
		10	294.0	152.6	153.0	153.0	153.2	154.0	155.6	**151.8**
		2	48.0	**14.6**	**14.6**	**14.6**	**14.6**	15.0	15.0	15.2
		4	53.3	**25.8**	**25.8**	**25.8**	**25.8**	26.6	26.4	27.2
250	50	6	62.8	**44.2**	**44.2**	**44.2**	**44.2**	45.2	44.6	45.4
		8	62.9	**64.8**	**64.8**	**64.8**	**64.8**	66.4	66.0	67.8
		10	68.5	82.8	82.8	82.8	82.8	84.0	**82.6**	85.4
		2	99.3	**18.0**	**18.0**	**18.0**	**18.0**	19.0	21.0	**18.0**
		4	135.9	**43.4**	**43.6**	**43.6**	**43.6**	46.2	46.2	45.6
250	100	6	170.5	**79.4**	**79.4**	**79.6**	**79.6**	81.4	80.8	81.2
		8	188.3	**111.4**	**111.4**	**111.4**	**111.4**	112.4	**111.4**	114.2
		10	240.1	**141.8**	**141.8**	**141.8**	**141.8**	143.6	142.2	144.4
		2	186.7	**26.6**	**26.8**	**27.0**	**27.0**	29.6	32.0	28.2
		4	221.2	**65.2**	**65.4**	**65.6**	**65.6**	68.8	71.2	68.4
250	150	6	268.3	**109.2**	**109.2**	**109.2**	**109.2**	110.4	112.4	112.6
		8	300.2	**147.6**	**147.6**	**147.8**	**147.8**	150.8	151.0	149.2
		10	347.5	**174.4**	**174.6**	**174.8**	**174.8**	177.2	176.0	175.2
		Total	**2261.4**	**2263.2**	**2264.6**	**2265.4**		2317.8	2318.2	2302.0
	*N*º of best values		27	27	27	27		0	3	3

Best values are in bold.

Table 13
Description of additional experiments

Experiment	Description
GA	Run plain BRKGA with chromosome adjustment
GA-M	Run plain BRKGA with chromosome adjustment and with the modified MOSP
GA-LS	Run plain BRKGA with chromosome adjustment and with the local search
GA-LS-M	Run plain BRKGA with chromosome adjustment and the local search and modified MOSP (i.e., BRKGA)

the number of item types chosen in the set $\{200, 250\}$, and the density (average number of item types per pattern) chosen in the set $\{2, 4, 6, 8, 10\}$, five instances were generated. As a postcondition, any instance where the customer graph can be divided into separate components is discarded to avoid relatively trivial instances. We compare BRKGA against AGA, DP2, and PMA using the results published in De Giovanni et al. (2013). DP2 and PMA were run on the same configuration as AGA by De Giovanni et al. (2013). In order to make the comparison as fair as possible and to take into account the nondeterministic nature of AGA and PMA (and the potentially long computational time of DP2 as an exact approach), De Giovanni et al. (2013) tested AGA five times for each instance and recorded the average computational time; then, DP2 and PMA were run and stopped after the same computational time, storing the best (if not provably optimal) solution found so far (PMA was repeated five times per instance).

Summary results are given in Table 12. Each row shows the results for each group of five instances with the same number of item types (I), number of patterns (P), and density (D). The columns corresponding to AGA and PMA report the minimum value of MOSP over the five runs. The column corresponding to DP2 reports the minimum value of MOSP obtained within the time limit. For the BRKGA, we include four columns /1, /2, /3, and /4 that correspond to the minimum value of MOSP obtained by BRKGA, over five runs, when we use as computational time the time limit used by AGA was divided by 1, 2, 3, and 4, respectively.

Table 12 shows that *BRKGA* consistently produces solutions that outperform the algorithms *AGA*, *PMA*, and *DP2* in 27 of all the 30 subgroups, even when a fourth of the computational time limit is used. A statistical analysis of the results using the Wilcoxon test for the matched pairs $BRKGA < AGA$, $BRKGA < PMA$, and $BRKGA < DP2$ shows that *BRKGA* is significantly better than all the other algorithms obtaining p-values for all paired comparisons smaller than 0.001.

Becceneri instances

In this subsection, we compare BRKGA against the low-order polynomial time *AS* heuristic for MOSP proposed by Ashikaga and Soma (2009). The comparison uses the Becceneri set of instances generated by Becceneri (1999), which covers the most common practical industrial scenario applications. The tests follow the notation used by Becceneri et al. (2004), that is, $n \times m$ that stands, respectively, for the number of items and number of patterns, and C is the maximum

Table 14
Computational results: Becceneri instances

I	*P*	*C*	Time		No. of generations										
n	*m*		(seconds)	AS	1	5	10	15	20	25	30	35	40	45	50
		5	0.001	6.90	6.30	6.30	6.30	6.30	6.30	6.30	6.30	6.30	6.30	6.30	6.30
10	10	6	0.002	7.80	7.70	7.70	7.70	7.70	7.70	7.70	7.70	7.70	7.70	7.70	7.70
		7	0.002	8.55	8.50	8.50	8.50	8.50	8.50	8.50	8.50	8.50	8.50	8.50	8.50
		8	0.001	9.10	9.10	9.10	9.10	9.10	9.10	9.10	9.10	9.10	9.10	9.10	9.10
		6	0.003	13.65	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50
		8	0.003	16.00	14.15	14.10	14.10	14.10	14.10	14.10	14.10	14.10	14.10	14.10	14.10
20	20	12	0.002	18.10	17.65	17.65	17.65	17.65	17.65	17.65	17.65	17.65	17.65	17.65	17.65
		14	0.002	18.80	18.80	18.80	18.80	18.80	18.80	18.80	18.80	18.80	18.80	18.80	18.80
		16	0.002	19.85	19.85	19.85	19.85	19.85	19.85	19.85	19.85	19.85	19.85	19.85	19.85
		8	0.007	22.40	19.50	19.25	19.15	19.15	19.15	19.15	19.15	19.15	19.15	19.15	19.15
		12	0.006	26.35	24.50	24.40	24.40	24.40	24.40	24.40	24.40	24.40	24.40	24.40	24.40
30	30	16	0.006	28.00	27.35	27.30	27.30	27.30	27.30	27.30	27.30	27.30	27.30	27.30	27.30
		20	0.005	29.30	29.20	29.20	29.20	29.20	29.20	29.20	29.20	29.20	29.20	29.20	29.20
		24	0.004	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00
		10	0.012	32.80	29.40	29.10	29.00	29.00	29.00	29.00	29.00	29.00	29.00	29.00	29.00
		15	0.012	36.80	34.50	34.30	34.30	34.30	34.30	34.20	34.20	34.20	34.20	34.20	34.20
		20	0.012	38.40	37.60	37.40	37.40	37.40	37.40	37.40	37.40	37.40	37.40	37.40	37.40
		25	0.012	39.30	39.30	39.30	39.20	39.20	39.20	39.20	39.20	39.20	39.20	39.20	39.20
		10	0.026	39.90	36.40	35.60	35.40	35.30	35.20	35.20	35.20	35.10	35.10	35.00	35.00
50	50	15	0.026	45.20	42.80	42.50	42.40	42.30	42.30	42.30	42.20	42.10	42.10	42.10	42.10
		20	0.022	47.10	46.10	45.70	45.70	45.70	45.70	45.70	45.70	45.70	45.70	45.70	45.70
		30	0.020	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30
		10	0.050	47.60	42.30	41.50	41.30	41.10	41.00	41.00	40.90	40.90	40.90	40.90	40.90
60	60	15	0.048	54.20	50.80	50.20	49.80	49.80	49.70	49.60	49.60	49.60	49.60	49.60	49.60
		20	0.048	56.80	55.70	55.00	55.00	55.00	55.00	55.00	55.00	55.00	55.00	55.00	55.00
		25	0.040	57.90	57.20	57.00	57.00	57.00	57.00	57.00	57.00	57.00	57.00	57.00	57.00
		10	0.301	77.70	69.45	67.85	66.55	65.90	65.55	65.35	65.20	65.10	65.10	65.05	65.05
		15	0.290	88.60	83.20	82.20	81.50	80.50	80.30	80.10	79.90	79.70	79.70	79.70	79.70
100	100	20	0.296	93.00	90.10	89.20	88.90	88.50	88.40	88.30	88.20	88.10	88.00	88.00	88.00
		30	0.296	97.00	95.90	95.70	95.60	95.50	95.40	95.40	95.40	95.40	95.40	95.40	95.40
		40	0.270	98.80	98.20	98.00	98.00	98.00	97.90	97.90	97.90	97.90	97.90	97.90	97.90
		10	1.388	114.60	103.25	100.45	99.15	98.00	96.95	96.35	96.10	96.00	95.95	95.95	95.95
		15	1.352	131.05	124.70	123.30	122.50	121.60	120.85	120.20	119.70	119.35	119.30	119.30	119.25
150	150	20	1.354	139.20	134.95	134.10	133.50	133.05	132.55	131.95	131.70	131.55	131.50	131.40	131.40
		25	1.322	142.60	140.75	139.95	139.40	138.85	138.40	138.20	138.00	137.90	137.90	137.90	137.90
		30	1.326	145.30	144.00	143.30	142.80	142.60	142.40	142.30	142.20	142.10	142.10	142.00	142.00
	Total MOSP			1928.0	1850.0	1834.6	1827.3	1821.5	1817.4	1814.5	1812.6	1811.2	1810.9	1810.6	1810.5
Percentage of improvement over AS					4.04	4.84	5.22	5.52	5.74	5.88	5.99	6.06	6.07	6.09	6.09

Table 15
Computational results: BRKGA component performance

I	*P*					(BRKGA)
n	*m*	*D*	GA	GA-M	GA-LS	GA-LS-M
		2	19.4	15.2	13.0	13.2
		4	35.8	31.6	29.8	29.8
200	50	6	52.6	49.2	48.4	48.0
		8	68.0	64.4	62.8	62.8
		10	83.2	80.2	79.0	78.6
		2	33.2	25.2	19.8	19.6
		4	60.8	53.8	50.6	50.0
200	100	6	94.6	88.2	84.0	82.8
		8	119.0	113.0	109.8	109.0
		10	141.8	134.6	133.4	132.0
		2	47.8	36.2	28.2	26.8
		4	85.6	75.6	69.0	67.4
200	150	6	122.8	113.0	108.2	107.0
		8	149.0	141.2	134.8	133.6
		10	163.0	159.4	154.8	152.6
		2	20.8	17.0	14.6	14.6
		4	32.2	27.8	26.0	25.8
250	50	6	50.0	45.8	44.2	44.2
		8	70.8	66.6	65.0	64.8
		10	89.0	84.6	82.8	82.8
		2	33.8	24.0	18.4	18.0
		4	58.6	48.8	44.6	43.6
250	100	6	90.8	84.4	80.8	79.4
		8	121.0	116.2	112.8	111.4
		10	150.2	144.8	143.0	141.8
		2	48.2	37.8	28.6	26.6
		4	83.2	75.0	66.8	65.2
250	150	6	125.8	118.0	111.4	109.2
		8	163.2	154.6	150.4	147.6
		10	189.4	180.4	176.8	174.6
Total MOSP			2603.6	2406.6	2291.8	2262.8
Percentage of improvement over GA				7.57	11.98	13.09

number of items allocated to a pattern. For each group, a sample of 20 random instances was generated.

In the test, we let BRKGA run once for 50 generations and registered the value of MOSP at the end of the first generation and then every five generations. Summary results are given in Table 14, where each row shows the results for the average MOSP values over the instances in the group. The column "Time" corresponds to the time used by BRKGA for one generation while the column *AS* represents the average MOSP obtained by the AS heuristic. The columns with headings 1, 5,...,50 correspond to the values of MOSP obtained by the BRKGA after the number of generations in the corresponding heading.

After only one generation the BRKGA obtains, for all the 36 groups, MOSP values that are equal to or better than the values obtained by the AS heuristic. The MOSP values obtained by BRKGA over all the groups on the first generation correspond already to an improvement of 4.04% over the values of the AS heuristic, and as the number of generations increase the improvement goes up to 6.09%. BRKGA always finds better values than AS in reasonable running time (often negligible). As the size of the instance increases, the computational time increases quite slowly.

Impact of each component on the BRKGA performance

To investigate the impact of each of the components included in BRKGA (GA, local search, and modified MOSP) on its performance, we conducted the additional experiments described in Table 13.

Each experiment was run five times for 100 generations on the 150 harder instances presented in Section "Harder instances." Summary results are given in Table 15. Each row shows the results for each group of five instances with the same number of item types (I), number of patterns (P), and density (D). The columns corresponding to GA, GA-M, GA-LS, and GA-LS-M report the minimum value of MOSP over the five runs.

From Table 15, it is clear that the plain BRKGA (GA) does not perform well since it produces an overall MOSP increase of 13% with respect to the full algorithm (BRKGA). The combinations of the plain BRKGA (GA) with the modified MOSP (GA-M) and with the local search (GA-LS) produce better results. Nevertheless, they are 5.5% and 1.1%, respectively, above the ones produced by the full BRKGA (GA-LS-M). Combining the plain BRKGA with both the local search and modified MOSP into GA-LS-M results in the best values of MOSP.

4. Concluding remarks

In this paper, we addressed the MOSP that consists of determining a sequence of cutting patterns that minimize the maximum number of open stacks during the cutting process. The approach proposed combines a *BRKGA* and local search procedure for generating the sequence of cutting patterns. A new fitness function for evaluating the quality of the solutions is also proposed. 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 more than 7000 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 partially supported by projects PTDC/EGE-GES/117692/2010 and NORTE-07-0124-FEDER-000057 funded by the North Portugal Regional Operational Programme (ON.2—O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF) and the Programme COMPETE; and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT). We would like to thank Luigi De Giovanni, Fernando Masanori Ashikaga, and Geoffrey Chu for sending us instances for the problem.

References

Ashikaga, F.M., Soma, N.Y., 2009. A heuristic for the minimization of open stacks problem. *Pesquisa Operacional* 29, 439–450.

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 padr oes para minimizaçao 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 Gent, I.P. (ed.) *Principles and Practice of Constraint Programming – CP 2009*, Vol. 5732 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 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, 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.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., 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.

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 for combinatorial optimization. *Journal of Heuristics* 17, 487–525.

Gonçalves, J.F., Resende, M.G.C., 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.C., 2013. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics* 145, 500–510.

Gonçalves, J.F., Resende, M.G.C., 2014. An extended Akers graphical with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research* 21, 215–246.

Gonçalves, 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.

Gonçalves, 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.

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

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

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. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 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. In Tinhofer, G., Mayr, E., Noltemeier, H., Syslo, M.M. (eds) *Computational Graph Theory*. Springer, Vienna, pp. 17–51.

Morán-Mirabal, L.F., González-Velarde, J.L., Resende, M.G.C., 2014. Randomized heuristics for the family traveling salesperson problem. *International Transactions in Operational Research* 21, 41–57.

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

Oliveira, A.C.M., Lorena, L.A.N., 2006. Pattern sequencing problems by clustering search. In Sichman, J.S., Coelho, H., Rezende, S.O. (eds) *Advances in Artificial Intelligence – IBERAMIA-SBIA 2006,* Vol. 4140 of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 218–227.

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

Spears, W.M., Dejong, K.A., 1991. On the virtues of parameterized uniform crossover. Proceedings of the Fourth International Conference on Genetic Algorithms, San Diego, CA, 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.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., Becceneri, J., Soma, N., 1999. Bounds for a problem of sequencing patterns. *Pesquisa Operacional* 19, 249–277.

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., 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., 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.