Computers & Industrial Engineering 112 (2017) 175–186

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Descent search approaches applied to the minimization of open stacks

Júnior Rhis Lima *, Marco Antonio M. Carvalho

Federal University of Ouro Preto, Morro do Cruzeiro, Ouro Preto, MG 35400-000, Brazil

ARTICLE INFO

Article history:
Received 1 December 2016
Received in revised form 8 August 2017
Accepted 13 August 2017
Available online 16 August 2017

Keywords:
Scheduling
Minimization of Open Stacks
Variable Neighborhood Descent

ABSTRACT

In this paper, new algorithms are proposed for solving the minimization of open stacks, an industrial cutting pattern sequencing problem. In the considered context, the objective is to minimize the use of intermediate storage, as well as the unnecessary handling of manufactured products. We introduce a new local search method, specifically tailored for this $\mathcal{NP}$-hard problem, which has wide practical applications. In order to further explore the solution space, we use this new local search as a component in two descent search methods associated with grouping strategies: variable neighborhood descent and steepest descent. Computational experiments were conducted involving 595 benchmark instances from five different sets through which the contributions of the proposed methods were compared with those of the state-of-the-art methods. The results demonstrate that the proposed algorithms are competitive and robust, as high quality solutions were consistently generated in a reasonable running time.

1. Introduction

In industrial contexts, the cutting stock problem constitutes a family of combinatorial optimization problems. These concern cutting larger units of raw material to produce smaller units (or *pieces*) satisfying some objective, such as trim loss minimization. The practical importance of such problems comes from industries that process raw materials, such as metal, wood, glass, and paper, in order to produce consumer goods. The set of pieces to be cut and their respective positions within the larger units define a *cutting pattern* (or just *pattern*).

When the cutting patterns have been defined, the raw material is ready to be processed. In the production environment considered, a single cutting machine processes a set P of different patterns to produce a set C of pieces according to specific demands. The production is divided into stages and it is considered that in each stage, a batch of a given different cutting pattern is processed. Therefore, all copies of a particular cutting pattern should be cut before a different cutting pattern is processed.

The sequence in which the different patterns are cut can affect different objectives. Those are commonly related to the use of intermediate storage, the pace of service for different purchase orders, and the homogeneity of the physical characteristics of the pieces produced. It is desirable then to determine the sequence of patterns to be processed according to such criteria, thus, defining the pattern sequencing problems.

Once produced, each piece is kept in a *stack* in which pieces of the same type are stored in the vicinity of the machine that produced it, i.e., in an intermediate storage of pieces. As long as there are pieces of a particular type to be produced, the associate stack is considered *open*. When the last piece of a type is produced, the associated stack is considered *closed* and can be removed from the production area to another location. Without loss of generality, we assume a physical limitation for storing stacks in the intermediate storage, such that all possible stacks cannot be simultaneously open. Therefore, it is mandatory to minimize the number of stacks simultaneously open by means of the patterns sequencing, characterizing the *minimization of open stacks problem* (MOSP), as described by Yuen (1991).

If the limit of the intermediate storage is reached, it becomes necessary to temporarily remove open stacks such that new stacks can be opened and the production is not suspended. The constant removal of open stacks requires the allocation of extra manpower, machinery, and time for the transportation and management of the stacks removed temporarily.

This paper presents three new approaches for solving the MOSP, based on different paradigms. The first is a tailored local search method designed to identify bottlenecks in a solution and attempt to modify it in order to minimize the maximum number of simultaneous open stacks. The second method consists of a *nested variable neighborhood descent* (NVND) metaheuristic implementation. The final proposed method consists of a *nested steepest descent* (NSD), which is a specialization of the previous method that

* Corresponding author.
E-mail addresses: juniorrhis1@gmail.com (J.R. Lima), mamc@iceb.ufop.br (M.A.M. Carvalho).

http://dx.doi.org/10.1016/j.cie.2017.08.016
0360-8352/

maintains the quality of the results and reduces the running time. Both NVND and NSD use the new local search, expanding its range of action, and grouping strategies, in order to reduce the search space.

Experiments were conducted using 595 real-world and artificial instances from the literature to evaluate the performance of the proposed descent methods using the state-of-the-art methods as a benchmark. The reported results are satisfactory: in most cases they showed that both the proposed methods were able to achieve the optimal solution values and demonstrated them to be competitive with the benchmark methods. Additionally, the running times are considered short, particularly those of the steepest descent method.

The remainder of this paper is organized as follows. Section 2 briefly presents the MOSP literature review; Section 3 describes the MOSP in detail; Section 4 describes the contributions of this work; Section 5 describes and discusses the computational experiments performed, and Section 6 presents the conclusions drawn from the study results and indicates directions for future research.

2. Literature review

Various constructive greedy heuristics were proposed by Yuen (1991) and further refined in Yuen (1995). Another greedy heuristic was also proposed by Faggioli and Bentivoglio (1998), but with an additional refinement step performed by a tabu search method. Later, Fink and Voß (1999) compared tabu search, simulated annealing, and classical constructive heuristics applied to the MOSP and to the minimization of order spread, a related problem. A number of hybrid metaheuristics were also proposed for the solution of the MOSP and related problems. These include *predatory search* (Linhares, 1999), *microcanonical optimization* (Linhares, Yanasse, & Torreão, 1999), constructive genetic algorithms (Oliveira & Lorena, 2002), multi-population evolutionary approaches (Mendes & Linhares, 2004), and clustering search (Oliveira & Lorena, 2006).

The *minimal cost node heuristic* (MCNh), which models the MOSP as a search for edges in MOSP graphs, was proposed by Becceneri, Yanasse, and Soma (2004) and is still considered the best performing ad hoc heuristic. Two constructive heuristics, based on the detection of Hamiltonian circuits and on collapse of edges on the MOSP graph were proposed by Ashikaga and Soma (2009). Later, Carvalho and Soma (2011) proposed two greedy heuristics derived from simplifications of the original methods of Ashikaga and Soma (2009) and de la Banda and Stuckey (2007). The simplification of the method of Ashikaga and Soma (2009) outperformed the original version.

De Giovanni, Massi, and Pezzella (2013a) addressed the MOSP using a genetic algorithm (GA), the fitness function of which used adaptive criteria defined upon the search space properties. De Giovanni et al. (2013b) considered the objective of minimizing the number of stages during which each stack remains open, in the context of very large scale integration (VLSI) design. The previously proposed GA was used to determine a threshold for the number of open stacks and a branch and cut algorithm was proposed to minimize the time during which the open stacks are open.

Another promising ad hoc heuristic, based on breadth-first search on MOSP graphs and a post-optimization step, was proposed by Carvalho and Soma (2015). Although faster and more robust than MCNh, it was considered less accurate.

Recently, Gonçalves, Resende, and Costa (2016) proposed a new objective function and the application of a parallel biased random-key GA (BRKGA) to the MOSP. The computational experiments reported indicate that this method was able to obtain optimal solutions for a comprehensive set of 6,141 benchmark instances taken from the literature. Currently, it is the state-of-the-art metaheuristic for the MOSP solution. Among the instances considered are those from the constraint modeling challenge (Smith & Gent, 2005), the Faggioli and Bentivoglio (1998) artificial instances, real-world VLSI design instances (Hu & Chen, 1990), and real-world woodcutting instances. These sets of instances were also considered in the computational experiments conducted in this study.

Exact approaches appear less frequently in the MOSP literature. Branch and bound formulations were proposed (Faggioli & Bentivoglio, 1998; Yanasse & Limeira, 2004; Yuen & Richardson, 1995), but without practical application because of runtime restrictions. In 2005, the MOSP was chosen as the subject of a constraint modeling challenge (Smith & Gent, 2005), which attracted the attention of the academic community. The winner of the challenge was the dynamic programming model of de la Banda and Stuckey (2007), which was able to solve almost all the 5,806 instances of that challenge. Later, Chu and Stuckey (2009) improved the previous method by employing a branch and bound strategy and new pruning rules. This method is currently the state-of-the-art exact method for the MOSP solution. Lopes and Carvalho (2015) proposed a linear programming model based on interval graphs; however, even for medium instances the model could not be run because of excessive memory consumption.

Important theoretical contributions include the modeling using graphs (Yanasse, 1997), bounds on the solution value (Yanasse, Becceneri, & Soma, 1999), the identification of special cases solved in deterministic polynomial time (Yanasse & Limeira, 2004), and different preprocessing operations (Yanasse & Senne, 2010). The latter also identified that a large part of the constraint modeling challenge instances was trivial or had a unique solution value. More recently, Sanches and Soma (2016) described a scheme for coding and solving $\mathcal{NP}$-hard and co-$\mathcal{NP}$-complete problems using DNA computing, including the MOSP.

Extensions of the MOSP include those that consider the number of open stacks as a constraint rather than in the objective function (Arbib, Marinelli, & Ventura, 2016; Lucero, Marenco, & Martínez, 2015) and those that address it simultaneously with the cutting stock problem (Arbib, Marinelli, & Pezzella, 2012; Matsumoto, Umetani, & Nagamochi, 2011; Yanasse & Lamosa, 2007). An interesting MOSP variation was reported in Hung, Chang, and Chen (2014), which considers a flow shop environment in the context of fabric cutting for apparel manufacturing. In this problem, the stacks move along the workstations of the production line; however, stacks are moved to the next workstation only after all of their pieces are processed in the previous workstation on the line.

3. Problem description

To illustrate the motivation of our study, we describe the MOSP applied in practice in the planning of a furniture manufacturing plant. In this context, large wood sheets, i.e., two-dimensional panels, are cut into smaller parts of different sizes by CNC machines. A given pattern represents a cutting diagram and the pieces are semifinished products, later assembled to produce finished goods.

A machine composed of different saws performs three-stage guillotine cuts, i.e., alternating horizontal and vertical edge-to-edge cuts, and has the capacity to cut hundreds of tons of wood per production day. After the cutting, each piece is routed to automatic unloading stations using conveyors. These unloading stations feed a local buffer for produced pieces available in the vicinity of the CNC machine. Each piece cut is stacked on a pallet that holds exactly one piece type on that buffer. The furniture manufacturing produces hundreds of different piece types, and each pattern may be composed of dozens of different piece types.

However, the local buffer has a limited capacity for storing pallets, such that a stack for each piece type cannot be simultaneously open on that buffer.

Fig. 1 depicts a small example with six different cutting patterns, numbered p_1 to p_6. Each piece type is numbered; the hatched area is the waste area of each pattern.

Let us consider a cutting machine having four unloading stations and a local buffer that has the capacity to store four pallets of pieces. Any pattern in this example can be cut without the need to relocate open stacks, because the largest number of piece types in a single pattern is exactly four (see pattern p_2). However, depending on the sequencing of patterns, the number of open stacks may exceed that which the local buffer is able to hold. In this case, a forklift is used to temporarily remove open stacks to an external storage area, freeing space for more open stacks. Fig. 2 illustrates the utilization of the local buffer and the external storage area at each production stage considering the sequence of patterns $[p_2, p_1, p_3, p_6, p_4, p_5]$. At each stage, the produced pieces are shown in white, and hatched pieces denote idle open stacks.

The pattern cut in the first stage, p_2, opens four stacks, associated with piece types 2, 4, 5, and 6, considering the buffer capacity. However, when pattern p_1 is cut at the second stage, a fifth stack needs to be opened to accommodate pieces of type 1. Piece types 5 and 6 are not cut at the second stage but, while idle, the associated stacks are still open. In order to make the production feasible, it is paused and the piece type 6 stack is temporarily removed to the external storage area so that a new stack can be opened in the local buffer for piece type 1. For the same reason, a second open stack needs to be removed from the local buffer at stage 3, when the maximum number of six simultaneous open stacks is reached. At stage 4, when the piece type 6 stack is needed again, it is brought back to the local buffer and substitutes the stack of piece type 2, which was closed. Stages 5 and 6 do not exceed the buffer capacity and the production is finished.

Formally, a MOSP instance is represented by a binary matrix $M = \{m_{ij}\}$ that associates the cutting patterns with the pieces. Table 1(a) presents the instance corresponding to Fig. 1. The set of pieces, numbered from 1 to 6, is represented by the rows and the set of cutting patterns, numbered from p_1 to p_6, is represented by the columns. An entry $m_{ij} = 1$ ($i \in C, j \in P$) indicates that the pattern p_j contains piece i. Otherwise, $m_{ij} = 0$. In the given example, pattern p_1 is composed of pieces 1, 2, and 4, pattern p_2 is composed of pieces 2, 4, 5, and 6, and so on.

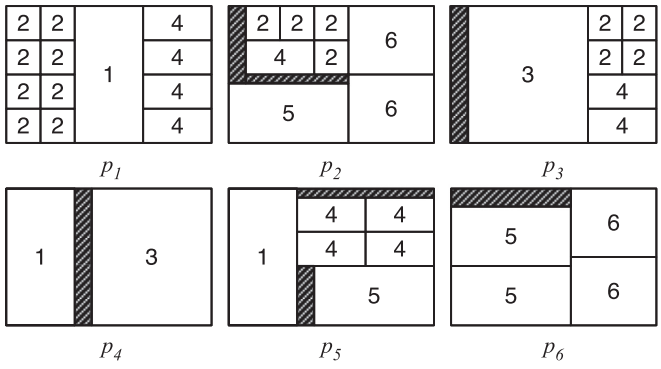

Fig. 1. Six cutting patterns.

A solution to the MOSP is a permutation π of matrix M columns, also represented by a permutation matrix $Q^\pi = \{q^\pi_{ij}\}$. This permutation matrix represents each production stage according to π and its elements are defined according to Eq. (1), where $\pi[n]$ denotes the processing order of the nth cutting pattern. Notice that Q^π holds the consecutive ones property, i.e., every entry between two nonzero entries is also considered a nonzero entry. If a piece, the corresponding stack of which is open, is not produced at a particular stage, this stack is counted as open if any pieces of that type remain to be processed. Thus, the consecutive ones property indicates that a stack remains open while idle. In other words, the stack is not closed because pieces still remain to be produced and therefore cannot be removed from the local buffer.

$$q^\pi_{ij} = \begin{cases} 1, & \text{if } \exists x, \exists y | \pi[x] \leqslant j \leqslant \pi[y] \text{ and } m_{ix} = m_{iy} = 1, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Parts (b) and (c) of Table 1 present two possible solutions for the input data given in (a), including the matricial representation of the solution illustrated in Fig. 2. The consecutive ones are represented by values 1 in boldface. For example, in part (b), third and fourth stages (in which the patterns p_3 and p_6 are processed), piece 1 is not produced because p_3 and p_6 do not contain that piece. However, the stack associated with piece 1 remains open until the last production stage, when pattern p_5 is produced.

The maximum number of simultaneous open stacks is given by the largest sum value of a column in Q^π considering the consecutive ones property. Such columns are called *bottlenecks* or *critical columns*. Eq. (2) presents the MOSP evaluation function, given an instance M and a solution π, from which Q^π is obtained.

$$Z^\pi_{MOSP}(M) = \max_{j \in P} \sum_{i=1}^{|C|} q^\pi_{ij} \quad (2)$$

The MOSP objective is to determine a sequencing π of products that minimizes the maximum number of simultaneous open stacks, as described in Eq. (3), where Π is the set of all $|P|!$ possible permutations.

$$\min_{\pi \in \Pi} Z^\pi_{MOSP}(M) \quad (3)$$

In the first sequencing, defined in Table 1(b), we have $\pi = [p_2, p_1, p_3, p_6, dep_4, p_5]$. In that sequencing, the maximum number of open stacks is six: in the third stage, stacks associated with all pieces will be simultaneously open. Table 1(c) presents a different solution, $\pi = [p_3, p_4, p_5, p_1, p_2, p_6]$, where the maximum number of open stacks is four, which occurs in the second, third, fourth, and fifth production stages.

Table 1
MOSP instance and two possible solutions for patterns processing.

	p_1	p_2	p_3	p_4	p_5	p_6		p_2	p_1	p_3	p_6	p_4	p_5		p_3	p_4	p_5	p_1	p_2	p_6
1	1	0	0	1	1	0	1	0	1	**1**	**1**	1	1	1	0	1	1	1	0	0
2	1	1	1	0	0	0	2	1	1	1	0	0	0	2	1	**1**	**1**	1	1	0
3	0	0	1	1	0	0	3	0	0	1	**1**	1	0	3	1	1	0	0	0	0
4	1	1	1	0	1	0	4	1	1	1	**1**	**1**	1	4	1	**1**	1	1	1	0
5	0	1	0	0	1	1	5	1	**1**	**1**	1	**1**	1	5	0	0	1	**1**	1	1
6	0	1	0	0	0	1	6	1	**1**	**1**	1	0	0	6	0	0	0	0	1	1
			(a)							(b)							(c)			

The consecutive ones are represented by values 1 in boldface.

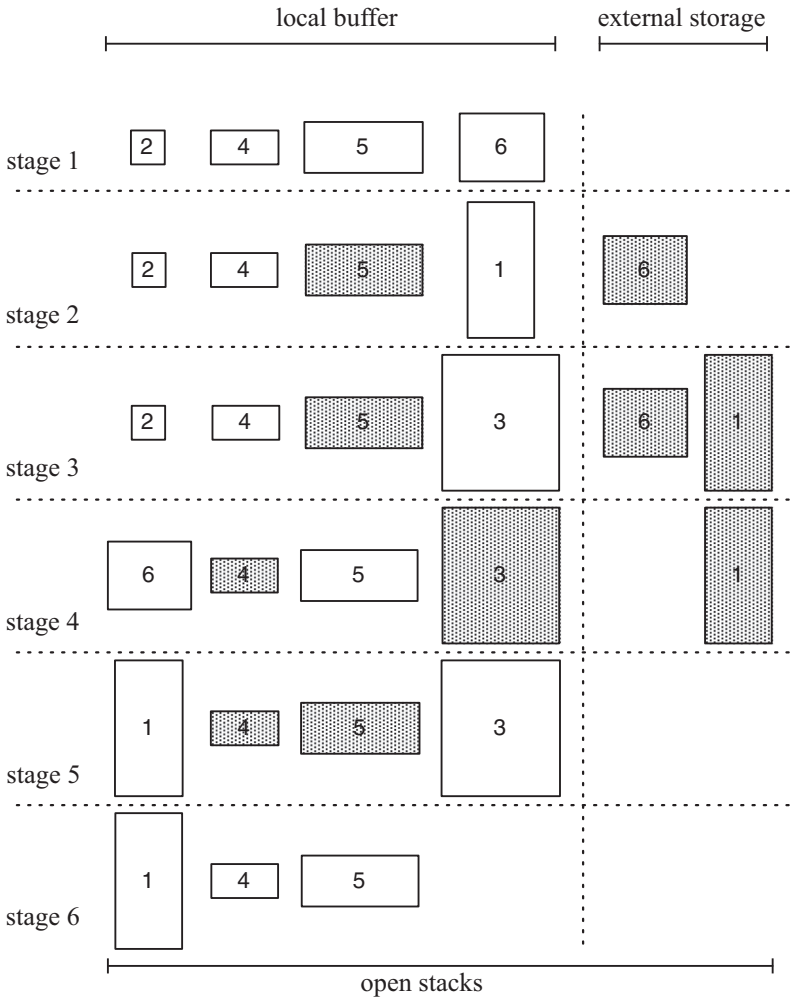

Fig. 2. Local buffer and external storage utilization at each production stage.

The MOSP complexity was proved $\mathcal{NP}$-hard according to the paper of Linhares and Yanasse (2002), which proved the equivalence between this problem and the modified cutwidth problem. In addition, the equivalence between MOSP and problems related to VLSI design problems, such as the gate matrix layout problem (GMLP) and programmable logic array folding problem, was established. In the same paper, a family of graph theory problems also equivalent to the MOSP was noted. These include interval thickness, node search game, edge search game, narrowness, split bandwidth, graph pathwidth, edge separation, and vertex separation.

4. Methods

In this section, we describe the details of each contribution of this paper: the new local search method and the proposed implementations of the NVND and NSD methods.

4.1. Initial solution

Using the new local search method, the NVND and NSD methods iteratively improve on an existing solution. We chose the ad hoc constructive heuristic described in Carvalho and Soma (2015) for generating initial solutions because of its robustness and simplicity.

This heuristic uses a dominance preprocessing procedure in order to reduce the number of patterns in the instances (Yanasse & Senne, 2010) and models the problem using MOSP graphs (Yanasse, 1997). In these graphs, vertices represent pieces and there is an edge between two vertices if and only if the corresponding pieces are found in a same pattern at least once. The MOSP graph is traversed by the well-known breadth-first search in order to generate a sequence of pieces, what provides useful insights into which stacks must be open simultaneously. The sequence of patterns is obtained from the sequence of pieces using the method of Becceneri et al. (2004). This greedy method attempts to avoid the unnecessary opening of stacks by sequencing a pattern only when all its associated stacks are already opened.

4.2. New local search method

The main idea of the proposed local search procedure, named *ReduceC1s*, is to identify a solution's bottleneck and attempt to reduce the occurrence of consecutive ones of the corresponding columns in M^{π}. Given an initial solution π, the new local search method identifies the set G of stacks (pieces) associated with the maximum number of open stacks, i.e., the stacks that represent the bottleneck of the problem. Then, patterns that contain any piece type in G are also identified. These patterns are removed from π and reinserted in a best insertion fashion, i.e., each pattern is inserted in π at the position that yields the best solution value. The priority of patterns in the reinsertion step is determined by their similarity to set G's composition: the larger the similarity value, the higher the priority.

Algorithm 1 presents the pseudocode for ReduceC1s. The input consists of an initial solution π, which is modified in the process. The first step is to find the set G of pieces in the bottleneck(s) of π (line 2). Then, the patterns that contain any pieces also in the bottleneck(s) are listed in L (line 3) and sorted in non-decreasing order of similarity to G (line 4). Each pattern p (lines 6–14) is removed from π (line 7), and reinserted at the position resulting in fewer consecutive ones in the bottleneck-related columns in M^{π} (line 8). The ReduceC1s procedure is applied while there is any improvement in the current solution (lines 9, 10, and 13).

Algorithm 1. ReduceC1s local search

Algorithm 1: ReduceC1s local search

input : solution π

1 **repeat**
2 insert into G the pieces contained in each bottleneck of π;
3 insert into L every pattern that contains any piece in G;
4 sort the elements of L according to the number pieces in common with G;
5 $improvement \leftarrow false$;
6 **foreach** *pattern* $p \in L$ **do**
7 remove p from π;
8 insert p in π at the position that yields the best solution value;
9 **if** π *was improved* **then**
10 $improvement \leftarrow true$;
11 **end**
12 **end**
13 **until** $improvement = false$;
14 **return** π;

An example of the proposed local search application follows. Considering the MOSP instance given in Table 2(a) and a solution $\pi = [p_2, p_1, p_3, p_5, p_4]$, represented in Table 2(b), the maximum number of simultaneous open stacks is four. The third stage is identified as the bottleneck, thus we have $G = \{1, 2, 3, 5\}$ and $L = \{p_3 p_1, p_5, p_4, p_2\}$. Consequently, these patterns in L are removed and inserted again in the solution, resulting in $\pi = [p_5, p_3, p_4, p_1, p_2]$. The new solution has a maximum of three simultaneous open stacks, as shown in Table 2(c).

Fig. 3 illustrates both solutions presented in Table 2(b) and (c). In the third stage in Fig. 3(a), the bottleneck of the solution has four open stacks, including one that was moved to the external storage while open. Fig. 3(b) presents the solution after applying the local search, where only the local buffer is used, meaning its capacity is considered in all the production stages and no additional handling of open stacks is needed.

In order to further explore the capability of the ReduceC1s local search, it was embedded in descent search strategies as a local search mechanism, as explained in the following sections.

4.3. Nested Variable Neighborhood Descent

The second contribution proposed in this work is an implementation of the variable neighborhood descent (VND) metaheuristic (Mladenović & Hansen, 1997). The main idea behind VND is that a local optimum shared by different neighborhood structures is more likely to be a global optimum than a local optimum of a single neighborhood structure. To this end, the method improves on an initial solution by successively applying different local search procedures, each one inducing a different neighborhood structure.

Table 2
Local search application example.

	p_1	p_2	p_3	p_4	p_5		p_2	p_1	p_3	p_5	p_4		p_5	p_3	p_4	p_1	p_2
1	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0
2	1	1	0	1	0	2	1	1	**1**	**1**	1	2	0	0	1	1	1
3	1	0	1	0	0	3	0	1	1	0	0	3	0	1	**1**	1	0
4	0	1	0	0	0	4	1	0	0	0	0	4	0	0	0	0	1
5	0	0	1	0	1	5	0	0	1	1	0	5	1	1	0	0	0
			(a)						(b)						(c)		

The consecutive ones are represented by values 1 in boldface.

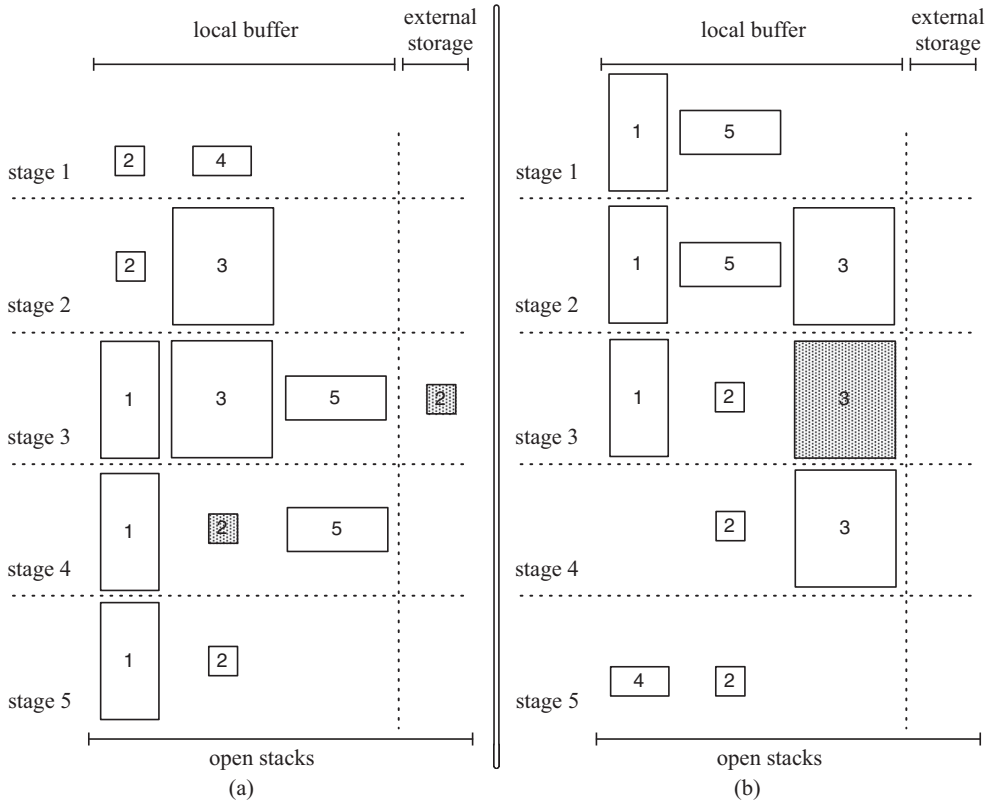

Fig. 3. Solution improvement by the proposed local search.

The VND efficiency lies in the definition of the neighborhood structures that explore different promising regions of the solution space. Our implementation is a *nested* VND (NVND), i.e., it explores a large neighborhood composed of different smaller neighborhoods (Hansen, Mladenović, Todosijević, & Hanafi, 2016). We considered two neighborhood structures: the neighborhood induced by the ReduceC1s local search procedure described in Section 4.2 and a set of *k-swap* neighborhoods, described below. The nesting of neighborhoods consists of finding a local optimum on a *k*-swap neighborhood and then applying the ReduceC1s local search procedure, moving the solution to another local optimum. If an improvement in the solution value is achieved, the current solution is updated and the *k*-swap neighborhood is explored again.

The *k*-swap local search procedure is a classical one concerning permutational problems. It consists of consecutively swapping the positions of *k* elements of a given solution, generating new solutions that differ in exactly *k* elements from the original one. All possible swaps are evaluated; however, only swaps that improve the solution value are performed, in a *first-improvement* fashion. When a swap has been performed, all possible swaps are evaluated again considering the updated solution. The *k*-swap method stops when no swap is able to improve the solution value. This local search method induces k_{max} different neighborhoods by varying the value of *k* in the range $[2, \ldots, k_{max}]$.

A complete exploration of the mentioned neighborhood structures for large instances may require a prohibitively long running time. To reduce the search space and the NVND running time, we employ a strategy of grouping consecutive patterns present in a solution. These groups are composed of γ distinct patterns at most. For example, let us consider an initial solution $\pi = [1, 3, 5, 4, 2]$. For $\gamma = 2$, the groups would be $[1, 3]$, $[5, 4]$, and $[2]$, and thus, groups of patterns are swapped instead of individual patterns; i.e., the *k*-swap method would operate over three elements, instead of five. After preliminary experiments, we chose to use $\gamma = 2$ and randomly select the pairs to be swapped, although all pairs are evaluated. Additionally, we chose to vary the value of the parameter *k* incrementally between 2 and 5 in the *k*-swap method. A benefit of this strategy is that it tends to retain part of the initial solution structure. The method for generating the initial solution provides good quality solutions, which we decided not to change significantly. As reported in Section 5, good results were obtained with this method.

During the tuning of algorithms, we learned that applying the *k*-swap local search after applying the ReduceC1s local search method achieved better solutions. We believe that the reason for this behavior is the complementarity of the methods; i.e., the combination of problem-specific moves and simple random moves provided a robust method.

Algorithm 2 presents the pseudocode for the NVND. The input consists of the original matrix *M*, an initial solution π, and parameters γ and k_{max}. While the solution value is improving (lines 1–17), the NVND is performed. Variables are initialized (lines 2 and 3), the current solution is divided into groups of size γ (line 4), and the neighborhoods are sequentially explored (lines 5–16). Each random neighbor solution from the *k*-swap neighborhood structure (line 6) is further explored by the ReduceC1s local search (line 7). If the solution value is improved (line 8), it is updated (line 9)

and divided into groups again (line 10), the improvement is recorded (line 11), and the *k*-swap neighborhood is explored from the start again (line 12). If no improvement is achieved, the *k*-swap local search moves to the next neighborhood (line 15).

Algorithm 2. Nested Variable Neighborhood Descent

```
Algorithm 2: Nested Variable Neighborhood Descent
   input : matrix M, initial solution π, group size γ, number of swap
           neighborhoods k_max.
 1 repeat
 2     k ← 1;
 3     improvement ← false;
 4     divide the elements of solution π into groups of size γ for k-swap use;
 5     repeat
 6         foreach solution π' obtained from π by a random k-swap move do
 7             apply the ReduceC1s local search in π';
 8             if π' is better than π then
 9                 update π with the contents of π';
10                 divide the new elements of π into groups of size γ;
11                 improvement ← true;
12                 go to line 2;
13             end
14         end
15         k ← k + 1;
16     until k = k_max;
17 until improvement = false;
18 return π;
```

Even after the adoption of the grouping strategy in the local search, the NVND running time may be prohibitive on very large instances because of the nested neighborhoods. This motivated a simplification of the proposed NVND implementation, described next.

4.4. Nested Steepest Descent algorithm

The third method proposed in this paper is an NSD algorithm. It comprises, given an initial solution, repeatedly applying local nested search procedures in a first-improvement fashion. This represents a simplification of the NVND described above: the *k*-swap neighborhoods are replaced by a single one, induced by the 2-swap method. By fixing the value of *k*, three of the previous swap neighborhoods are not considered. Additionally, the pattern grouping strategy is replaced by a *sliding window* strategy.

Given a solution π, a *window* is a subsequence of ω consecutive patterns ($1 \leqslant \omega \leqslant |P|$). Initially, this is the first subsequence ω patterns in π, i.e., $\pi[1 \ldots \omega]$. This window then slides a stage to the right, i.e., $\pi[2..\omega+1]$, and so on until all the patterns in π have been considered. For example, for $\pi = [1, 3, 5, 4, 2]$ and $\omega = 2$, the windows are [1, 3], [3, 5], [5, 4], and [4, 2]. Therefore, in this method, the 2-swap procedure is applied to two different windows that do not share patterns, randomly selected, as a local search procedure. After preliminary experiments, we chose $\omega = 2$.

The nesting of local search procedures is performed as described previously: for each neighbor solution generated by the 2-swap method, the ReduceC1s local search, described in Section 4.2, is applied. If the resulting solution has a better value than the current one, it is updated. The NSD method is applied as long as the generated solutions are improving.

Algorithm 3 presents the pseudocode for the proposed NSD method. The input consists of the original matrix *M*, an initial solution π, and parameter ω. Initially, all windows of size ω are generated (line 2) for use in the 2-swap local search. While the solution value is improving (lines 2–13), the NSD is performed. Each random neighbor solution from the 2-swap local neighborhood structure (line 4) is further explored by the ReduceC1s local search (line 5). If the solution value is improved (line 6), it is updated (line 7), new windows are generated (line 8), the improvement is recorded (line 9), and the new 2-swap neighborhood is explored from the start (line 10). If no improvement is achieved, the NSD stops.

Algorithm 3. Nested Steepest Descent

Algorithm 3: Nested Steepest Descent

input : matrix M, initial solution π, window size ω.

1 generate all windows of size ω from π for 2-swap use;
2 **repeat**
3 *improvement* $\leftarrow$ *false;*
4 **foreach** *solution* π' *obtained from* π *by a random 2-swap move* **do**
5 apply the ReduceC1s local search in π';
6 **if** π' *is better than* π **then**
7 update π with the contents of π';
8 generate all windows of size ω from the new π;
9 *improvement* $\leftarrow$ *true*;
10 go to line 3;
11 **end**
12 **end**
13 **until** $improvement = false$;
14 **return** π;

Although the sliding window strategy may resemble the pattern grouping employed by the proposed NVND, these are different strategies. Using the sliding window allows the same pattern to appear in different windows, increasing the search space. However, this allows the basic structure of the initial solution to be preserved. As mentioned previously, we chose not to change the initial solutions significantly.

5. Computational experiments

The proposed methods were implemented in C++ and compiled using the g++ 4.4.1 compiler with the -O3 compilation flag. The experiments were conducted on a 3.2 GHz Intel i5 Quad Core computer with 16 GB of RAM under the Ubuntu 12.4.1 operational system.

The maximum number of simultaneous open stacks, the gap between the solution and the optimal solution values, the standard deviation, and the average running times were analyzed in this experiment. To this end, five sets of real-world and artificial benchmark instances from the literature are considered, which are separately described the following sections. The proposed methods are compared to the state-of-the-art methods for solving the MOSP: the branch and bound of Chu and Stuckey (2009), the BRKGA of Gonçalves et al. (2016), and the MCNh of Becceneri et al. (2004).

The optimal results for these instances were established by the branch and bound of Chu and Stuckey (2009). The same results were also found by the BRKGA for the first four sets of instances (there are no reported results for last set), and consequently, the same analyses apply to both methods. A fair comparison of running times is not possible because the methods were run on different architectures. Nevertheless, according to the original reports, the BRKGA running times are negligible, and the branch and bound running times are low for small instances, but grow fast as the size of instances increases. The MCNh was run on the same computational setup as the proposed methods; thus, a fair running time comparison is possible.

The following tables present, for each instance, the number of patterns ($|P|$), the number of pieces ($|C|$), and the optimal solution values (OPT) found by the branch and bound and the BRKGA (except for the last set of instances). For the MCNh, the tables present the solution values (S) and runtime (T), in seconds. For the proposed NVND and NSD methods, the tables also present the best solution values (S^*), the average runtime (T) expressed in seconds, and the standard deviation (σ) over 20 independent runs. Bold values identify solutions that reached the optimal values. Where mentioned, the gap (or percentage distance) between the solution and the optimal values, is calculated as $100 \times (\text{obtainedvalue} - \text{optimalvalue})/\text{optimalvalue}$.

5.1. Real-world minimization of open stacks problem instances

This set, provided by the *Sheet Cutting and Process Optimization for Furniture Industry* (SCOOP) Consortium,[1] contains 187 real-world MOSP instances from two European woodcutting companies. However, most of these instances are too small, e.g., $|P| = |C| = 2$, implying trivial solutions. Therefore, 24 instances with significant dimensions, ranging from 10 pieces and patterns to 49 pieces and 134 patterns, were selected for the experiments. Table 3 presents the results for these instances.

Both NSD and NVND achieved good solutions for this set of instances in short running times. The NSD method (a gap of 1.61% and 87.50% of optimal solutions found) performed better than the NVND method (a gap of 2.69% and 79.17% of optimal solutions found). In fact, NSD and NVND failed to achieve the optimal solutions for three and five instances, respectively. In these cases, both methods opened a single additional stack; thus, the minimum error from the optimal solution obtained by the branch and bound and by the BRKGA. MCNh delivered the worst solution values (a gap of 25.29% and 33.33% of optimal solutions found), despite its negligible running times, lower than 0.002 s for any instance. The average standard deviation over 20 independent runs is low for both proposed methods. NSD achieved standard deviation equal zero for 12 instances and NVND did the same for 9 instances.

[1] Available at http://www.scoop-project.net.

Table 3
Results for the SCOOP instances results.

Instance	$\|P\|$	$\|C\|$	*OPT*	*MCNh*		*NVND*			*NSD*		
				T	*S*	*T*	S^*	σ	*T*	S^*	σ
A_FA+AA-1	37	107	12	0.00	16	1.02	**12**	0.32	0.28	**12**	0.79
A_FA+AA-11	28	99	11	0.00	15	0.11	**11**	0.45	0.08	**11**	0.65
A_FA+AA-12	20	75	9	0.00	13	0.01	**9**	0.45	0.02	**9**	0.00
A_FA+AA-13	37	134	17	0.00	25	1.69	18	0.50	0.40	18	0.60
A_FA+AA-15	18	68	9	0.00	10	0.01	**9**	0.23	0.01	10	0.00
A_FA+AA-2	19	75	11	0.00	13	0.00	**11**	0.50	0.01	**11**	0.37
A_FA+AA-6	21	79	13	0.00	18	0.02	**13**	0.51	0.04	**13**	0.47
A_FA+AA-8	28	82	11	0.00	16	0.12	12	0.48	0.08	12	0.51
A_AP-9.d-3	16	20	6	0.00	**6**	0.00	**6**	0.23	0.00	**6**	0.00
A_AP-9.d-10	13	20	6	0.00	7	0.00	**6**	0.51	0.00	**6**	0.47
A_AP-9.d-11	21	27	6	0.00	8	0.00	**6**	0.00	0.00	**6**	0.00
A_AP-9.d-6	20	31	5	0.00	6	0.00	**5**	0.00	0.00	**5**	0.00
B_12F18-11	15	21	6	0.00	7	0.00	**6**	0.51	0.00	**6**	0.00
B_12M18-12	22	31	6	0.00	7	0.00	**6**	0.45	0.00	**6**	0.50
B_18AB1-32	11	15	6	0.00	**6**	0.00	7	0.00	0.00	**6**	0.00
B_18CR1-33	18	20	4	0.00	**4**	0.00	**4**	0.00	0.00	**4**	0.00
B_22X18-50	11	14	10	0.00	**10**	0.00	**10**	0.00	0.00	**10**	0.00
B_23B25-52	21	29	5	0.00	8	0.00	**5**	0.00	0.00	**5**	0.00
B_39Q18-82	10	14	5	0.00	**5**	0.00	6	0.00	0.00	**5**	0.22
B_42F22-93	10	18	5	0.00	**5**	0.00	**5**	0.00	0.00	**5**	0.00
B_CARLET-137	12	14	5	0.00	**5**	0.00	6	0.00	0.00	**5**	0.41
B_CUC28A-138	26	37	6	0.00	7	0.00	**6**	0.51	0.00	**6**	0.50
B_GTM18A-139	20	24	5	0.00	**5**	0.00	**5**	0.23	0.00	**5**	0.00
B_REVAL-145	49	60	7	0.00	11	0.97	**7**	0.45	0.16	**7**	0.31
Average			7.75	0.00	9.71	0.16	7.96	0.26	0.05	7.88	0.24

Bold values identify solutions that reached the optimal values.

Table 4
Results for Hu and Chen (1990) instances.

Instance	$\|P\|$	$\|C\|$	*OPT*	*MCNh*		*NVND*			*NSD*		
				T	*S*	*T*	S^*	σ	*T*	S^*	σ
v1	8	6	3	0.00	**3**	0.00	4	0.00	0.00	4	0.00
v4000	17	10	5	0.00	6	0.00	**5**	0.00	0.00	**5**	0.00
v4050	16	13	5	0.00	6	0.00	**5**	0.49	0.00	**5**	0.00
v4090	27	23	10	0.00	**10**	0.00	**10**	0.00	0.00	**10**	0.00
v4470	47	37	9	0.00	14	1.32	**9**	0.22	0.00	10	0.59
vc1	25	15	9	0.00	**9**	0.00	**9**	0.47	0.00	**9**	0.00
vw1	7	5	4	0.00	**4**	0.00	**4**	0.00	0.00	**4**	0.00
vw2	8	8	5	0.00	**5**	0.00	**5**	0.00	0.00	**5**	0.00
w1	21	18	4	0.00	6	0.00	**4**	0.44	0.00	**4**	0.37
w2	33	48	14	0.00	15	0.00	**14**	0.22	0.02	**14**	0.52
w3	70	84	18	0.00	22	20.87	**18**	0.57	2.40	**18**	0.46
w4	141	202	27	0.00	36	4070.19	**27**	0.52	35.50	28	2.19
wan	7	9	6	0.00	**6**	0.00	**6**	0.00	0.00	**6**	0.00
wli	10	11	4	0.00	**4**	0.00	**4**	0.00	0.00	**4**	0.00
wsn	25	17	8	0.00	**8**	0.00	**8**	0.00	0.00	**8**	0.00
x0	48	40	11	0.00	**11**	1.11	**11**	0.37	0.35	**11**	0.49
x1	10	5	5	0.00	**5**	0.00	**5**	0.00	0.00	**5**	0.00
x2	15	6	6	0.00	**6**	0.00	**6**	0.00	0.00	**6**	0.00
x3	21	7	7	0.00	**7**	0.00	**7**	0.00	0.01	**7**	0.00
x4	8	12	2	0.00	**2**	0.00	**2**	0.00	0.00	**2**	0.00
x5	16	24	2	0.00	**2**	0.00	**2**	0.00	0.00	**2**	0.00
x6	32	49	2	0.00	**2**	0.00	**2**	0.00	0.00	**2**	0.00
x7	8	7	4	0.00	**4**	0.00	**4**	0.00	0.00	**4**	0.00
x8	16	11	4	0.00	**4**	0.00	**4**	0.00	0.00	**4**	0.00
x9	32	19	4	0.00	**4**	0.06	**4**	0.00	0.03	**4**	0.00
Average			7.12	0.00	8.04	163.74	7.16	0.13	1.53	7.24	0.18

Bold values identify solutions that reached the optimal values.

The results suggest that the proposed local search is able to generate good solutions independently of the other neighborhoods chosen, and did not strongly influence the running time.

5.2. Real-world very large scale integration design instances

This set contains 25 instances of the GMLP, a VLSI design problem equivalent to the MOSP, as mentioned in Section 1. The GMLP is also represented by a binary matrix holding the consecutive ones property, the columns of which represent logical gates and the rows the nets connecting a subset of the gates through wires. A wire may cross gates that are not part of a net, if necessary. The problem asks for a permutation of gates that minimizes the maximum number of overlapping wires from different nets, such that the number of physical tracks of the printed circuit is also minimized. These real-world instances, introduced by Hu and Chen (1990), were obtained from Asian companies. The results for this set are shown in Table 4.

Table 5
Results for Faggioli and Bentivoglio (1998) instances.

$\|P\|$	$\|C\|$	*OPT*	*MCNh*		*NVND*			*NSD*		
			T	*S*	*T*	S^*	σ	*T*	S^*	σ
10	10	5.50	0.00	5.60	0.00	5.80	0.00	0.00	**5.50**	0.00
10	20	6.20	0.00	7.30	0.00	6.50	0.36	0.00	**6.20**	0.29
10	30	6.10	0.00	8.30	0.00	6.60	0.41	0.00	6.30	0.18
10	40	7.70	0.00	9.20	0.00	7.80	0.24	0.00	7.80	0.06
10	50	8.20	0.00	9.80	0.00	**8.20**	0.25	0.00	**8.20**	0.18
15	10	6.60	0.00	**6.60**	0.00	**6.60**	0.17	0.00	**6.60**	0.03
15	20	7.20	0.00	8.60	0.00	7.40	0.41	0.00	7.40	0.26
15	30	7.30	0.00	9.00	0.00	7.80	0.36	0.00	7.60	0.34
15	40	7.20	0.00	9.80	0.00	7.30	0.33	0.01	**7.20**	0.30
15	50	7.40	0.00	10.40	0.00	**7.40**	0.32	0.00	7.50	0.26
20	10	7.50	0.00	7.70	0.00	**7.50**	0.16	0.00	**7.50**	0.00
20	20	8.50	0.00	8.70	0.00	**8.50**	0.26	0.01	**8.50**	0.15
20	30	8.80	0.00	10.00	0.01	9.00	0.41	0.02	9.00	0.33
20	40	8.50	0.00	10.70	0.01	8.80	0.43	0.02	8.60	0.43
20	50	7.90	0.00	10.90	0.01	8.20	0.35	0.02	8.10	0.35
25	10	8.00	0.00	**8.00**	0.00	**8.00**	0.00	0.00	**8.00**	0.00
25	20	9.80	0.00	10.50	0.02	**9.80**	0.32	0.02	**9.80**	0.39
25	30	10.50	0.00	11.90	0.04	10.80	0.42	0.04	10.60	0.44
25	40	10.30	0.00	12.50	0.04	10.60	0.33	0.05	10.40	0.45
25	50	10.00	0.00	12.20	0.05	10.20	0.38	0.05	10.20	0.42
30	10	7.80	0.00	**7.80**	0.00	**7.80**	0.00	0.00	**7.80**	0.00
30	20	11.10	0.00	11.50	0.05	11.20	0.14	0.05	11.20	0.26
30	30	12.20	0.00	13.30	0.14	**12.20**	0.33	0.08	12.30	0.43
30	40	12.10	0.00	14.20	0.19	**12.10**	0.38	0.10	**12.10**	0.48
30	50	11.20	0.00	13.90	0.19	11.70	0.28	0.11	11.40	0.47
40	10	8.40	0.00	**8.40**	0.00	**8.40**	0.00	0.01	**8.40**	0.00
40	20	13.00	0.00	13.60	0.31	**13.00**	0.15	0.12	13.20	0.18
40	30	14.50	0.00	15.30	0.88	14.70	0.21	0.24	14.70	0.39
40	40	14.90	0.00	16.60	1.23	15.50	0.14	0.32	15.10	0.50
40	50	14.60	0.00	16.80	1.40	15.10	0.35	0.35	15.30	0.47
Average		9.30	0.00	10.64	0.15	9.48	0.26	0.05	9.42	0.27

Bold values identify solutions that reached the optimal values.

Again, similar solutions were generated, despite the neighborhood structure being adopted. In this second set of instances, the NVND (96% of optimal solutions found and a gap of 0.56%) performed better than NSD (88% of optimal solutions found and a gap of 1.69%). In fact, the NVND was not able to find the optimal solution for only one instance, while the NSD failed to find optimal solutions for three instances. Again, the branch and bound and the BRKGA were able to find all the optimal solutions for this set. Both NVND and NSD outperformed MCNh (72.00% of optimal solutions found and a gap of 12.92%) in terms of solution quality. Once more, MCNh generated solutions in a very short running time.

The running times of NVND increased considerably faster than those of NSD, particularly in larger instances, such as *w*3 and *w*4. This is explained by the larger search spaces of the *k*-swap neighborhoods explored. These long running times, although not prohibitive, motivated us to implement the NSD, as an attempt to reduce the computation effort required to solve large instances. Nonetheless, NSD presented a large standard deviation for the *w*4 instance: 2.19 stacks over 20 independent runs. For all other instances, NSD generated solutions with low standard deviation in a short, frequently negligible, running time.

5.3. Artificial minimization of open stacks problem instances of Faggioli and Bentivoglio (1998)

The third set, proposed by Faggioli and Bentivoglio (1998), contains 300 small and medium-sized artificial MOSP instances and a relatively wide variety of dimensions. The problems are grouped into subsets of 10 instances each, according to the number of patterns and pieces. Table 5 presents the average results for this set.

Both NVND and NSD were able to achieve comparable high quality solutions in short running times, although NSD performed slightly better. The NVND method found 82.00% of the optimal solutions (246 instances) and obtained a gap of 1.97%. The NSD method found 88.67% of the optimal solutions (266 instances) and obtained a gap of only 1.25%. The methods also reached all the optimal solution values for some groups in all independent runs. Together with the low average standard deviation (0.26 and 0.27 for NVND and NSD, respectively), these indicators demonstrate robustness and competitiveness with the state-of-the-art methods. MCNh, however, delivered a poor performance (a gap of 14.41% and 36.67% of optimal solutions found) as compared to the other methods. This method was able to find optimal solutions for instances with fewer pieces (e.g., $|C| = 10$) and failed in most other cases.

5.4. First constraint modeling challenge artificial minimization of open stacks problem instances (Smith & Gent, 2005)

The fourth set has 46 selected artificial MOSP instances from the original set proposed for the *First Constraint Modeling Challenge*[2] (Smith & Gent, 2005). These instances were selected considering their dimensions and the absence of special structures that facilitate the solution, as analyzed by Yanasse and Senne (2010). Except for *Shaw* (a group of 25 instances), all other instances are individual. The results are shown in Table 6, which has an additional column, *n*, identifying the number of instances. The average solution values and running times are reported for the whole group of *Shaw*

[2] Available at https://ipg.host.cs.st-andrews.ac.uk/challenge/.

Table 6
Results for selected instances of the *First Constraint Modeling Challenge* (Smith & Gent, 2005).

Instance	$\|P\|$	$\|C\|$	n	OPT	$MCNh$		$NVND$			NSD		
					T	S	T	S^*	σ	T	S^*	σ
NWRS1	20	10	1	3	0.00	**3**	0.00	**3**	0.00	0.00	**3**	0.00
NWRS2	20	10	1	4	0.00	**4**	0.00	5	0.22	0.00	**4**	0.00
NWRS3	25	15	1	7	0.00	**7**	0.00	**7**	0.50	0.00	**7**	0.44
NWRS4	25	15	1	7	0.00	**7**	0.00	**7**	0.00	0.00	**7**	0.00
NWRS5	20	20	1	12	0.00	**12**	0.01	**12**	0.00	0.02	**12**	0.00
NWRS6	30	20	1	12	0.00	**12**	0.02	**12**	0.00	0.04	**12**	0.00
NWRS7	60	25	1	10	0.00	**10**	0.17	**10**	0.47	0.12	**10**	0.51
NWRS8	60	25	1	16	0.00	**16**	0.98	**16**	0.00	0.26	**16**	0.00
GP1	50	50	1	45	0.00	**45**	3.66	**45**	0.00	0.64	**45**	0.00
GP2	50	50	1	40	0.00	**40**	11.00	**40**	0.37	1.66	**40**	0.60
GP3	50	50	1	40	0.00	**40**	10.77	41	0.00	2.93	**40**	0.00
GP4	50	50	1	30	0.00	**30**	1.74	31	0.00	0.60	31	0.00
GP5	100	100	1	95	0.00	96	4,709.22	**95**	0.00	64.28	**95**	0.00
GP6	100	100	1	75	0.00	**75**	2,975.64	77	0.00	74.63	76	0.00
GP7	100	100	1	75	0.00	**75**	5,114.71	77	0.00	78.17	76	0.37
GP8	100	100	1	60	0.00	**60**	4,977.36	62	0.22	83.73	61	0.22
Shaw	20	20	25	13.68	0.00	14.00	0.01	**13.68**	0.21	0.01	**13.68**	0.17
Miller	40	20	1	13	0.00	**13**	0.67	**13**	0.37	0.19	**13**	0.00
SP1	25	25	1	9	0.00	**9**	0.00	**9**	0.31	0.01	**9**	0.22
SP2	50	50	1	19	0.00	23	2.54	20	0.00	0.52	20	0.50
SP3	75	75	1	34	0.00	37	177.93	**34**	0.32	5.89	35	0.59
SP4	100	100	1	53	0.00	57	1,671.34	**53**	0.37	26.31	54	1.05
Average				21.76	0.00	22.20	427.34	21.98	0.07	7.39	21.91	0.10

Bold values identify solutions that reached the optimal values.

Table 7
Results for Chu and Stuckey (2009) instances.

$\|P\|$	$\|C\|$	OPT	$MCNh$		$NVND$			NSD		
			T	S	T	S^*	σ	T	S^*	σ
30	30	19.60	0.00	20.48	0.07	19.64	0.23	0.08	19.68	0.24
40	40	25.48	0.00	26.8	0.60	25.56	0.31	0.35	25.64	0.39
50	50	31.40	0.00	32.8	2.82	31.68	0.32	1.08	31.76	0.46
50	100	38.44	0.00	39.4	82.99	38.64	0.19	11.06	39.04	0.32
75	75	44.84	0.00	47.72	50.42	45.40	0.30	6.89	45.52	0.62
100	50	41.08	0.00	47.72	5.69	41.76	0.33	2.03	41.44	0.79
100	100	58.20	0.00	61.84	339.66	59.20	0.35	26.60	59.36	0.83
125	125	70.20	0.00	76.4	1563.22	71.32	0.37	76.78	71.96	1.02
Average		41.16	0.00	44.15	255.68	41.65	0.30	15.61	41.80	0.58

instances. At the bottom of the table, the average values considering all 46 instances are shown.

The NVND and NSD methods presented similar values for gap (0.90% and 0.70%, respectively – the best mark for NSD) and rate of optimal solutions found (86.96% and 84.78%, respectively). Each of the two methods delivered its lowest average standard deviation on this set (0.07 and 0.10, respectively, for NVND and NSD). In fact, each method was able to generate solutions with zero standard deviation for almost half of the instances, including the largest ones. This indicates a robust performance, and, considered in conjunction with the rate of optimal solutions found, indicates a low divergence from the solutions generated by the branch and bound and the BRKGA. MCNh delivered its best gap (2.02%) in this set of instances. Although only 41.30% of the optimal solutions was found, the divergence from the optimal solutions was also controlled. The noteworthy running times were under 0.003 s, even for some of the largest instances considered, e.g., $|P| = |C| = 100$.

The NVND running times increased quickly and were the longest in all sets of instances, exceeding one hour for those with 100 patterns and pieces, e.g., the *GP* instances. However, the NSD running time increased more slowly, remaining under 90 s. Again, the proposed local search helped to generate comparable solutions from both swap neighborhood structures.

5.5. Large artificial minimization of open stacks problem instances of Chu and Stuckey (2009)

The fifth and final set consists of 200 artificial MOSP instances, proposed by Chu and Stuckey (2009). These instances were randomly generated, with dimensions ranging from 30 to 125 patterns and pieces. For each different dimension, 5 instances were generated varying the fixed number of nonzero elements per columns between 2, 4, 6, 8, and 10. In contrast to the previous set, none of these instances is decomposable or has any other structure that facilitates its solution. There are no reported BRGKA solutions for this set of instances, and thus, the comparisons do not consider this method. Table 7 presents the average solution values and running times; each row corresponds to 25 instances.

In this set of larger instances, the NVND (51.00%), the NSD (53.50%), and the MCNh (22.00%) methods achieved their worst rate of optimal solutions. Additionally, NVND and NSD presented a larger average standard deviation (0.30 and 0.58, respectively). However, these values are still considered low, as is the gap of both proposed methods: 1.52% for the NVND and 1.57% for the NSD. Notwithstanding the decrease in accuracy for this set of instances, both proposed methods consistently approximate the solutions generated by the state-of-the-art methods, proving them to be

competitive. Once more, MCNh (a gap of 7.06%) was outperformed by both proposed methods, despite its impressive running times, under 0.004 s.

As this set contains the largest instances, one could expect again a quick increase in the running times and relatively long running times. However, although the densities of these instances are varied, the larger instances are less dense than the others. Therefore, the NVND and NSD running times did not grow as fast as in the second and fourth sets of instances. For instances with 100 or 125 pieces and patterns, the NVND running time was between 5.5 min and approximately 26 min. The NSD running time was between 26 s and approximately 1.2 min, the largest incurred by any instance in the experiments. The NVND running times, although longer, are completely acceptable in real-world contexts.

The reported indicators demonstrate that both proposed methods consistently delivered near optimal solutions and that their performance is robust, as the divergence from the optimal solutions is controlled. It becomes clear that the proposed local search is able to improve the solutions further, regardless of the size of the search space induced by the swap neighborhood structures, as it produces high quality solutions.

6. Conclusions

We have proposed a new local search method and two systematic search methods for solving the MOSP. The computational experiments involved several benchmark instances taken from the literature and state-of-the-art methods. We emphasize the following: (*i*) the substantial rates of optimal solutions found; (*ii*) the low divergence from the optimal solution values; and (*iii*) the robustness of the methods. Future research include embeddeding the NSD method in a *matheuristic*, or in one of the branch and bound methods presented in the literature, as it provides good quality upper bounds in a short running time.

Acknowledgements

This research was supported by the National Counsel of Technological and Scientific Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq) and the Minas Gerais State Agency for Research and Development (Fundação de Amparo à Pesquisa do Estado de Minas Gerais, FAPEMIG).

References

Arbib, C., Marinelli, F., & Pezzella, F. (2012). An LP-based tabu search for batch scheduling in a cutting process with finite buffers. *International Journal of Production Economics, 136*(2), 287–296.

Arbib, C., Marinelli, F., & Ventura, P. (2016). One-dimensional cutting stock with a limited number of open stacks: Bounds and solutions from a new integer linear programming model. *International Transactions in Operational Research, 23*(1–2), 47–63.

Ashikaga, F. M., & Soma, N. Y. (2009). A heuristic for the minimization of open stacks problem. *Pesquisa Operacional, 29*(2), 439–450.

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*(14), 2315–2332.

Carvalho, M. A. M., & Soma, N. Y. (2011). Simplified methods for the minimization of open stacks problem. *Gestão & Produção, 18*(2), 299–310.

Carvalho, M. A. M., & Soma, N. Y. (2015). A breadth-first search applied to the minimization of the open stacks. *Journal of the Operational Research Society, 66*(6), 936–946.

Chu, G., & Stuckey, P. J. (2009). Minimizing the maximum number of open stacks by customer search. In *Proceedings.... Lecture notes in computer science. Association for constraint programming* (Vol. 5732, pp. 242–257). Berlin: Springer.

De Giovanni, L., Massi, G., & Pezzella, F. (2013a). An adaptive genetic algorithm for large-size open stack problems. *International Journal of Production Research, 51*(3), 682–697.

De Giovanni, L., Massi, G., Pezzella, F., Pfetsch, M., Rinaldi, G., & Ventura, P. (2013b). A heuristic and an exact method for the gate matrix connection cost minimization problem. *International Transactions in Operational Research, 20*(5), 627–643.

de la Banda, M. G., & Stuckey, P. J. (2007). Dynamic programming to minimize the maximum number of open stacks. *INFORMS Journal on Computing, 19*(4), 607–617.

Faggioli, E., & Bentivoglio, C. A. (1998). Heuristic and exact methods for the cutting sequencing problem. *European Journal of Operational Research, 110*(3), 564–575.

Fink, A., & Voß, S. (1999). Applications of modern heuristic search methods to pattern sequencing problems. *Computers & Operations Research, 26*(1), 17–34.

Gonçalves, J. F., Resende, M. G. C., & Costa, M. D. (2016). A biased random-key genetic algorithm for the minimization of open stacks problem. *International Transactions in Operational Research, 23*(1–2), 25–46.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2016). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*.

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*(8), 836–845. Aug.

Hung, Y.-F., Chang, C.-Y., & Chen, H. (2014). Determining ideal fabric cutting times for apparel manufacturing by using mixed integer programming and a heuristic method. *European Journal of Industrial Engineering, 8*(6), 814–835.

Linhares, A. (1999). Synthesizing a predatory search strategy for vlsi layouts. *IEEE Transactions on Evolutionary Computation, 3*(2), 147–152.

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

Linhares, A., & Yanasse, H. H. (2002). Connections between cutting-pattern sequencing, vlsi design, and flexible machines. *Computers & Operations Research, 29*(12), 1759–1772. Oct.

Lopes, I. C., & Carvalho, J. V. A. d. (2015). Graph properties of minimization of open stacks problems and a new integer programming model. *Pesquisa Operacional, 35*, 213–250.

Lucero, S., Marenco, J., & Martínez, F. (2015). An integer programming approach for the 2-schemes strip cutting problem with a sequencing constraint. *Optimization and Engineering, 16*(3), 605–632.

Matsumoto, K., Umetani, S., & Nagamochi, H. (2011). On the one-dimensional stock cutting problem in the paper tube industry. *Journal of Scheduling, 14*(3), 281–290.

Mendes, A., & Linhares, A. (2004). A multiple-population evolutionary approach to gate matrix layout. *International Journal of Systems Science, 35*(1), 13–23.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*(11), 1097–1100.

Oliveira, A., & Lorena, L. (2002). A constructive genetic algorithm for gate matrix layout problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 21*(8), 969–974.

Oliveira, A. C. M., & Lorena, L. A. N. (2006). Pattern sequencing problems by clustering search. In *Proceedings of the 18th Brazilian conference on advances in artificial intelligence (IBERAMIA-SBIA'06)* (pp. 218–227). Berlin, Heidelberg: Springer-Verlag.

Sanches, C., & Soma, N. (2016). A general resolution of intractable problems in polynomial time through dna computing. *Biosystems, 150*, 119–131.

Smith, B., & Gent, I. (2005). Constraint modeling challenge. In B. Smith, & I. Gent (Eds.), *The fifth workshop on modelling and solving problems with constraints*. IJCAI, Edinburgh, Scotland.

Yanasse, H., Becceneri, J., & Soma, N. (1999). Bounds for a problem of sequencing patterns. *Pesquisa Operacional, 19*(2), 249–277.

Yanasse, H., & Limeira, M. (2004). Refinements on an enumeration scheme for solving a pattern sequencing problem. *International Transactions in Operational Research, 11*(3), 277–292.

Yanasse, H. H. (1997). On a pattern sequencing problem to minimize the maximum number of open stacks. *European Journal of Operational Research, 100*(3), 454–463.

Yanasse, H. H., & Lamosa, M. J. P. (2007). An integrated cutting stock and sequencing problem. *European Journal of Operational Research, 183*(3), 1353–1370.

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*(3), 559–567.

Yuen, B. J. (1991). Heuristics for sequencing cutting patterns. *European Journal of Operational Research, 55*(2), 183–190.

Yuen, B. J. (1995). Improved heuristics for sequencing cutting patterns. *European Journal of Operational Research, 87*(1), 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*(3), 590–598.