versão impressa ISSN 0101-7438 / versão online ISSN 1678-5142

A GENETIC SYMBIOTIC ALGORITHM APPLIED TO THE ONE-DIMENSIONAL CUTTING STOCK PROBLEM

Rodrigo Rabello Golfeto
Escola de Engenharia Industrial Metalúrgica
Universidade Federal Fluminense (UFF)
Volta Redonda – RJ, Brasil

Antonio Carlos Moretti
Inst. de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas (UNICAMP)
Campinas – SP, Brasil

Luiz Leduíno de Salles Neto*
Departamento de Ciência e Tecnologia
Universidade Federal de São Paulo (UNIFESP)
São José dos Campos – SP, Brasil
luiz.leduino@unifesp.br

* *Corresponding author* / autor para quem as correspondências devem ser encaminhadas

Recebido em 08/2007; aceito em 04/2009 após 2 revisões
Received August 2007; accepted April 2009 after 2 revisions

Abstract

This work presents a genetic symbiotic algorithm to minimize the number of objects and the setup in a one-dimensional cutting stock problem. The algorithm implemented can generate combinations of ordered lengths of stock (the cutting pattern) and, at the same time, the frequency of the cutting patterns, through a symbiotic process between two distinct populations, solutions and cutting patterns. Working with two objectives in the fitness function and with a symbiotic relationship between the two populations, we obtained positive results when compared with other methods described in the literature.

Keywords: cutting stock problem; genetic algorithm; symbiosis.

Resumo

Neste trabalho desenvolvemos um algoritmo genético simbiótico com objetivo de minimizar o número de objetos processados e o *setup* num problema de corte unidimensional. Nosso algoritmo genético gera seus próprios padrões em conjunto com soluções para o problema, através de um processo simbiótico entre duas populações distintas, a de soluções e a de padrões. Trabalhando com os dois objetivos na função de aptidão e com a relação simbiótica entre as duas populações, obtivemos resultados competitivos em relação aos métodos descritos na literatura.

Palavras-chave: problema de corte de estoque; algoritmo genético; simbiose.

1. Introduction

The cutting stock problem (CSP) is a classic problem in the area of Operations Research. In one-dimensional cases, it can be defined as the problem of finding the best way of cutting ordered items from a stock roll of length L such that trim loss is minimized and the total order is satisfied. This is a common problem arising in the production of paper (Haeesler, 1976; Diegel, 1988), steel (Eilon, 1960; Wäscher *et al.*, 1985), windows (Stadler, 1990), etc. A CSP can be broken down into two distinct sub-problems: (1) the generation of the cutting pattern and (2) the determination of its frequency in the final solution.

The Cutting Stock Problem is one of the first applications of Operations Research methods; it was first studied by Kantorovich in the thirties. Problems of the same nature were dealt with by Paull & Walter (1954), Metzger (1958) and Eilon (1960). However, as observed by Dowsland & Dowsland (1992), the methods used at that time were only appropriate for small problems. The papers of Gilmore & Gomory (1961, 1963) produced a great impact in this area by proposing an efficient method to work with a large scale cutting stock problems through the use of column generation procedures. Haessler's (1975) was the first work to treat the nonlinear cutting stock problem, considering the reduction of trim loss and the number of setups of the cutting machine.

Genetic Algorithms (GAs) were introduced by Holland (1975) and, in the last decade, they became a promising method for finding good solutions to optimization problems. GAs are a specific class of evolutionary computation (EC) as they employ techniques inspired by the theory of evolution, which was developed in 1859 by Darwin, who tried to establish a logical outline of the changes in a population's inherited traits from generation to generation. Several papers attempt to simulate some of the biological aspects, such as predator-prey, sexual recombination and so on. In our case, we subdivide the CSP into two sub-problems in order to simulate the symbiotic relationship, whereby two or more distinct populations have a collaborative relationship.

Genetic applications of the CSP generally employ cutting patterns generated by other methods, such as Branch-and-Bound, Constructive Heuristics with Branch-and-Bound, Constructive Heuristics with Random Search, Residual Heuristic with Branch-and-Bound, and Residual Heuristic with Random Search. Next, the frequency of each pattern is calculated (Boleta *et al.*, 2005; Khalifa *et al.*, 2006). Liang *et al.* (2002) propose an evolutionary programming algorithm (Fogel, 1995) with only one swap mutation operator that, according to the authors, outperforms GAs for one-dimensional CSPs with and without contiguity. However, the above studies do not aim at minimizing the setup cost. In the present study, our objective is to minimize the costs of setup and of processed raw material using a genetic algorithm – applied to a symbiotic relationship – that generates the cutting pattern together with its frequency.

In Section 2, we describe a mathematical model of the cutting stock problem for minimizing the number of objects and setups. In Section 3, we briefly present the main concepts of genetic algorithms. Section 4 is dedicated to the symbiosis process. In Section 5, we describe the applications. Finally, in Sections 6 and 7, we exhibit our calculations and final conclusions, respectively.

2. The One-Dimensional Cutting Stock Problem

The Standard Cutting Stock Problem is characterized by cutting stock rolls of width W (called *objects*) into smaller rolls of width w_i (where $W > w_i$), aiming at satisfying the demand d_i for each item i, for $i = 1,2,...,m$. According to Dyckhoff's typology (1990), this problem is classified as 1/V/I/R.

Each combination of items in an object is called the cutting pattern, and each change in the cutting pattern has a cutting-machine setup cost. The number of setups is of great importance when we have to process a large order in a short period. Wäscher (1990) criticizes the traditional planning models for cutting stock problems; he says they do not consider all of the process factors that tend to affect a company's profit, such as the setup number or the amount of stockpiled material created during the various cuts. Consequently, we have to find the balancing point between the number of setups and the quantity of objects used.

The mathematical model for minimizing the number of objects and setups can be stated as:

$$\begin{aligned} \text{Minimize} \quad & c_1 \sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \delta(x_j) \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \qquad i = 1,...,m. \\ & x_j \in \mathrm{N} \qquad\qquad j = 1,...,n. \end{aligned}$$

where a_{ij} is the number of items i in cutting pattern j; x_j is the number of objects processed with cutting pattern j; n is the number of the cutting pattern and

- c_1 is the cost for each master roll (object);
- c_2 the cost of setup and $\delta(x_j) = \begin{cases} 1 & \text{if } x_j > 0, \\ 0 & \text{if } x_j = 0. \end{cases}$.

We compare our approach with five methods, that also work with the two objective: setup and number of processed objects:

- **SHP**: This method was proposed by Haessler (1975). It is based on a repeat-pattern exhaustion technique (Pierce, 1964), where the cutting patterns are determined successively and, for each iteration, some aspiration parameters have to be satisfied. These aspiration parameters control the trade-off between the two costs, c_1 and c_2.
- **Kombi234**: This method was developed by Foester & Wäscher (2000). It is based on combining cutting patterns to reduce the number of setups for a given solution; it's an extension of the work of Diegel *et al.* (1993). This method maintains the number of processed objects constant; in other words, the sum of frequencies for each pattern is kept equal for all iterations. Once the final solution is found, the number of patterns can be reduced by up to 60% of the original number.
- **NANLCP**: This method, developed by Moretti & Salles Neto (2008), minimizes the number of objects and setups through the application of the Lagrangian function. This method also uses an adaptation of the column-generation method proposed by Gilmore & Gomory (1961, 1963) for non-linear models and a simple rounding heuristic to obtain an integer solution.

- **Hybrid Heuristic:** This method, proposed by Yanasse & Limeira (2006), is a hybrid procedure consisting of three phases. In the first phase, patterns are generated and the "good" ones are selected and used to reduce the problem; in the second phase, the reduced problem is solved and, in the third phase, a pattern-reduction technique is applied. The authors argue that the computational tests performed indicate that the proposed scheme provides alternative solutions to the pattern-reduction problem that are not overcome by other solutions obtained using procedures previously suggested in the literature.
- **ILS:** Umetani *et al.* (2006) presented a local search algorithm that uses two types of local search: (1) the 1-add neighborhood and (2) the shift neighborhood. Linear programming techniques were added to local search procedures to reduce the number of solutions for each neighborhood and improve performance. A sensitivity analysis technique was introduced to solve the large number of associated LP problems quickly. Umetani *et al.* (2006) compared ILS to Kombi234, to SHP, and to an exact branch-and-price method (BP) suggested by Belov & Scheithauer (2000), who proposed a method similar to the work of Vanderbeck (2000), yet with few variables. Vanderbeck (2000) investigates the problem of minimizing a number of different cutting patterns as in nonlinear integer programming, where the number of objects is fixed and determined after solving the problem using the Gilmore-Gomory strategy. In his article, Vanderbeck uses a Dantzig-Wolfe breakdown method that he extended of Vanderbeck (1999) to solving the resulting integer-programming problem. Umetani *et al.* (2006) claim that the ILS algorithm obtains better solutions than those obtained by the SHP, KOMBI234 and BP approaches.

3. Genetic Algorithms

According to Von Zuben (2003), genetic algorithms (GAs) are based on the works of Holland (1962), Bremermann (1962) and Fraser (1957). However, the genetic algorithm was actually introduced only in Holland's work (1975). In the last decade, it became a promising method for discovering solutions to optimization problems.

GAs are a specific class of evolutionary computation (EC), utilizing techniques inspired by the theory of evolution – such as natural selection, whereby stronger creatures have greater chances of reproducing and introducing their characteristics into the next generation – as a problem-solving paradigm.

The following important concepts are associated with genetic algorithms:

- Fitness: an individual's level of adaptability to his environment.
- Genes: functional blocks of DNA.
- Genome: an individual's genetic pattern.
- Selection: the mechanism responsible for selecting a population's top individuals for reproduction; the following are the most common types of selection: competition, elitism, and diversity – or a combination of various types.
- Crossover: also called recombination, crossover is the genetic operator used to combine two individuals to generate a new individual. There are many types of crossovers, the most common being One-point, Two-point and Uniform crossovers.
- Mutation: the probability of one gene mutating; this operator is implemented after the crossover.

4. Symbiosis

According to Allaby (1998), symbiosis is a general term describing a situation in which dissimilar organisms live together in close association. As originally defined, the term embraces all types of mutual and parasitic relationships. Its current use is often limited to mutually beneficial species-interaction. Mutualism is defined as an interaction between members of two different species, with benefits to both. Pianka (1994) gives some examples of interactions between species, such as nectar-feeding birds and flowering plants; ants and plants; and birds and buffalos. In Table 1, we can observe the benefit or harm to each individual in a symbiotic relationship.

Table 1 – Impact of symbiotic relationships on organisms.

Relationship	Self	Opponent
Amensalism	Neutral	Harm
Commensalism	Benefit	Neutral
Competition	Harm	Harm
Mutualism	Benefit	Benefit
Parasitism	Benefit	Harm
Predation	Benefit	Harm
Proto-cooperation	Benefit	Benefit

The process of *symbiogenesis* was introduced in (Watson & Pollack 1999) as the genesis of new species via the genetic integration of symbionts. For example, eukaryote cells (from which all plants and animals descend) have a symbiogenic origin. This can occur when the symbionts have a high degree of association.

The works of Eguchi *et al.* (2003), Hirasawa *et al.* (2000) and Mao *et al.* (2000) also simulate symbiosis relationships. However, in their algorithms, each individual of the population is treated as a different species that develops a symbiotic relationship with another individual. Hirasawa *et al.* (2003) give one example of a possible relationship: *"if individual i exists near individual j and the fitness of individual i is greater than that of individual j, then individual i exploits individual j"*.

The genetic symbiotic algorithm (GSA), also called cooperative algorithm (Potter, 1997; Kim *et al.* 2000), basically breaks the problem down into n sub-problems using n different species. Dividing the problem into n distinct populations, we can solve the problem utilizing simple structures that, working together, can be more powerful than complex structures. Kim *et al.* (2001, 2006) propose an endosymbiotic evolutionary algorithm for optimization; the basic idea is to incorporate eukaryotic cell evolution (Margullis, 1981) into the existing symbiotic algorithms. Under this approach, when an individual meets a highly fit partner, the whole combination evolves for some time without changing the partner.

Tsujimura *et al.* (2001) presented a symbiotic genetic algorithm for job shop scheduling. Chang *et al.* (2002) presented a symbiotic evolutionary algorithm for dynamic facility layout problems. In the next section, we explain the GSA's application to the CSP, aiming at minimizing raw material and setup costs. We have no knowledge of any other study that applies the GSA to the CSP.

5. Application

When applied to the CSP, the main difference between the GSA and the classical genetic algorithm (CGA) is its ability to construct cutting patterns regardless of the solution, thus eliminating (replacing) inefficient or unused pattern-population cutting patterns. Another interesting advantage GSAs have over CGAs is that the patterns are not part of the solution; although the solution depends on the pattern, we can work with them separately.

Khalifa *et al.* (2006) solve the CSP using a genetic algorithm whereby the genes of each solution are processed in pairs and the first gene of each solution represents the frequency of the pattern represented by the second gene. In our application, the second gene represents an individual from the pattern population. Figure 1 shows patterns 37, 11 and 32 with their respective frequencies: 2, 4 and 5.

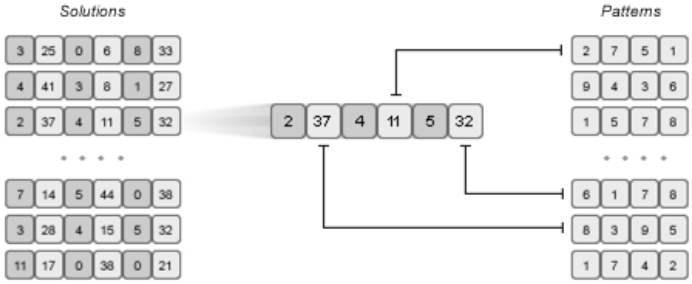

Figure 1 – Gene Structure.

We believe mutualism is the biological relationship that is the most relevant to our application, since both individuals benefit from the relationship; in this case, the relationship presents great adaptability to the environment.

In the next subsections, we explain the structure of each population. At this point, it is worth mentioning that we call the individuals of the first population *solutions* and the individuals of the second population *patterns*.

In the case of our implementation, we considered three stopping criteria:

- Maximum time of execution: 500 seconds;
- Maximum number of generations: 10,000 generations;
- Convergence: if the algorithm cannot find a better solution in 500 generations, it stops.

5.1 Individual Solutions

Below, we describe the parameters, obtained in an experimental way, of the first population (that is, *solutions*):

- Population size: 1,000 individuals;
- Type of selection: elitism, 70% of the best individuals;
- Crossover rate: 30%;
- Crossover type: uniform, 70% chance for the best individual (see figure 2);
- Mutation rate: we calculate the probability of 2 genes mutating; that is, if the individual has k genes, the probability of each gene mutating is $2/k$ (see figure 3).

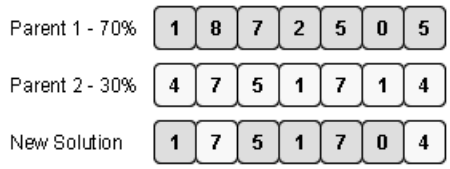

Figure 2 – Uniform crossover.

Figure 3 – Mutation of an individual/solution.

To determine the size of the DNA chain (i.e., the number of genes of an individual) in our application, we need to estimate the maximum number of setups that one problem could have. We chose *m* to represent this number, since the problem of minimizing the number of objects and setups used in a cutting plan can be written as a linear programming problem, whereby each constraint represents an item's demands. The number of an individual's genes is fixed as twice the maximum number of setups. However, if the biggest item in a cutting plan is less than or equal to half the size of an object, the following procedure is adopted:

```
If (Quantity of Items > 30) Then
    Gs = 32
Otherwise If (Quantity of Items > 15) Then
    Gs = 24
Otherwise
    Gs = 16
End-If
```

where Gs is the number of genes of an individual. We adopted this strategy, seeking to obtain, already in the first generations, solutions with a small number of different cutting patterns.

Represented by the odd genes, the frequency of each pattern has an upper and lower limit. This is done to restrict the problem's search region. The lower limit is set to zero, and the upper limit is defined as the largest order in the cutting plan. This is done to produce a pattern containing just one type of item. The fitness function for individual *i*, which contains the numbers of both the processed objects and the setups, is defined as

$$F_s(i) = c_1 \sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \delta(x_j) + \tau + \rho$$

where

- t_j is the trim loss for pattern j
- τ is the relative trim loss; that is, $\tau = \dfrac{\sum_{j=1}^{n} t_j x_j}{W \sum_{j=1}^{n} x_j}$

- mr is the number of items that are not present in the solution
- dr_i is the residual demand for item i; that is, $dr_i = d_i - \sum_{j=1}^{n} a_{ij} x_j$
- $\rho = 10^4 \times mr + 10^6 \times \sum_{i=1}^{m} dr_i$ is the penalty incurred if the solution is not feasible.

The costs c_1 and c_2 are used explicitly and the parameter τ has two important functions: (1) it is used to measure how good a local minimum is; a little change in the relative trim loss will provide an improvement in fitness, extending the application of the method; (2) it is a comparison factor; when we have two or more equally suitable solutions, we choose only one for the elite list. The elite list contains the 700 most suitable individuals (70% of the population's best individuals).

Additionally, at every 100 generations we randomly generate 200 individuals and place them on the elite list, replacing the 200 worst solutions. We adopt this strategy and the τ parameter in order to increase the population's diversity.

5.2 Individual Patterns

Pattern-population parameters, also obtained in an experimental way, are significantly different when compared to solution-population parameters:

- Population size: 600 individuals;
- Type of selection: elitism, 66% of the best individuals;
- Crossover rate: 34%;
- Crossover type: 2 points (see figure 4);
- Mutation rate: 90% chance of one gene mutating (see figure 5).

Figure 4 shows how the crossover in the pattern population is accomplished.

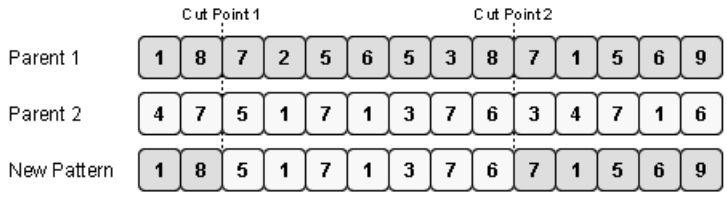

Figure 4 – Two-point crossover.

Figure 5 shows how the mutation in the pattern populations is accomplished. The gene is deactivated when its corresponding item doesn't fit in the pattern, and activated otherwise. For instance, the fifth gene, corresponding to the item 5, was deactivated before the mutation, because if it made part of the pattern, the sum of the items would be larger than the size of the object. As after the mutation the fourth gene changed from 2 to 3, and the item 3, unlike item 2, didn't fit in the pattern, therefore, we have a space to activated item 5.

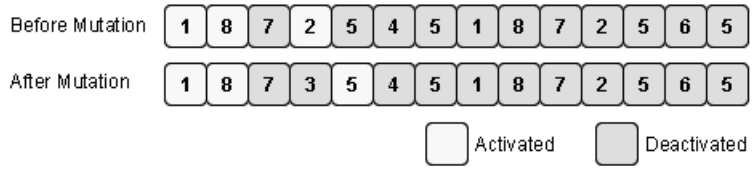

Figure 5 – Mutation of an individual/pattern.

The length of the DNA chain is equal to the largest integer that is smaller or equal to the size of the master roll divided by the size of the smallest item. By doing so, we can guarantee that it is possible to create a pattern using only one type of item. For example, in the pattern of the Figures 4 and 5, the DNA chain has 14 genes. This means that the smallest item has its size w_i satisfying $14 = \left\lceil W / w_i \right\rceil$.

However, some patterns will not use all the genes available in the DNA chain since we generate items in the pattern as long as the sum of their lengths does not exceed the size of the master roll. That is, we insert one item into the pattern from left to right, having the DNA chain as a reference, if and only if the pattern has enough space to accommodate the item.

We calculate the fitness of each pattern based on the solution population's elite list. For each individual/solution, we add points to the fitness of the individual pattern considered in the solution. We calculate a pattern's fitness according to the following rule:

```
For i = 1 to n_EliteSolutions
    For each pattern j in solution i DO
        FitnessPattern(j) = FitnessPattern(j) + 1 + (1 / i)
    End For
End For
```

where *n_EliteSolutions* is the number of individuals/solutions in the elite list; the first individual/solution in the list is the most suitable, the second individual/solution is the second most suitable, and so on.

This way, we give priority to the cutting pattern that appears in the most suitable solutions. During the evolutionary process, these suitable patterns should generate other suitable patterns that, through their symbiotic relationship with the individuals/solutions, will result in better solutions.

5.3 Pseudo-Code

The steps below describe how to obtain a solution for the CSP that minimizes the number of processed objects and setups at the same time:

Step 1: Initialize the population of solutions with random values

Step 2: Initialize the population of patterns with random values

Step 3: Calculate the fitness of individuals/solutions

Step 4: Select the solutions with the greatest fitness

Step 5: Add points to the fitness of the patterns of elite solutions

Step 6: Select the patterns with the greatest fitness

Step 7: Use the crossover and mutation operators to generate new solutions

Step 8: Use the crossover and mutation operators to generate new patterns

Step 9: If any stop criterion has been satisfied, then *Stop*; otherwise return to *Step 3*

6. Computational Experiments

In order to evaluate our method, we solved 1800 problems generated by CUTGEN1 (as proposed by Gau & Wäscher (1995)), divided into 18 classes, with the same seed and parameters used by Foester & Wäscher (2000). In terms of solution quality, we compared our approach to the following heuristics: SHP, proposed by Haessler (1975); Kombi234, developed by Foester & Wäscher (2000); NANLCP, proposed by Salles Neto & Moretti (2008); ILS, proposed by Umetani *et al.* (2006); and the Hybrid Heuristic, proposed by Yanasse & Limeira (2006).

Table 2 describes the parameters used for each class. We generated six classes with small items (classes 1 to 6), six classes with diverse items (classes 7 to 12), and six classes with large items (classes 13 to 18).

Table 2 – The parameters associated with the 18 classes.

Class	v1	v2	m	d
1	0.01	0.20	10	10
2	0.01	0.20	10	100
3	0.01	0.20	20	10
4	0.01	0.20	20	100
5	0.01	0.20	40	10
6	0.01	0.20	40	100
7	0.01	0.80	10	10
8	0.01	0.80	10	100
9	0.01	0.80	20	10
10	0.01	0.80	20	100
11	0.01	0.80	40	10
12	0.01	0.80	40	100
13	0.20	0.80	10	10
14	0.20	0.80	10	100
15	0.20	0.80	20	10
16	0.20	0.80	20	100
17	0.20	0.80	40	10
18	0.20	0.80	40	100

The method was implemented in FORTRAN 90/95, using the Microsoft FORTRAN Power Station compiler, in a PC AMD SEMPRON 2300+ computer (1,5 MHz/640MB RAM) with a windows operating system. The source code is available at *http://www.otimizacao.net.* The NANLCP and SHP methods were implemented in FORTRAN using a PC AMD ATHLONXP

computer (1,800 MHz/512MB RAM) with a Linux operating system. The Kombi234 results were obtained from Foester & Wäscher's study (2000); they implemented their code in MODULA-2, using MS-DOS 6.0 on an IBM 486/66. The ILS heuristic was encoded in C language and was run on an IBM-compatible personal computer (Pentium IV 2 GHz/1 GB memory) using the Linux OS. The Hybrid Heuristic (HH) was performed using C++ on an Intel Celeron microcomputer (266 MHz/128MB RAM) and a Sun Ultra 30 workstation (296MHz/384MB RAM).

The method achieved convergence of results in all 1800 problems; that is, the algorithm terminated its calculations before reaching 10,000 generations or 500 seconds.

In Table 3, we present the results for three different setup costs, $c_2 = \{1,5,10\}$, respectively labeled Symbio1, Symbio5 and Symbio10. Table 3 presents the average setup number for the SHP, Kombi234, ILS, NANLCP, HH and Symbio methods in each class. The ILS method works with an upper limit on the optimal value f_{opt} of the number of processed objects, which is calculated using a branch-and-price algorithm based on the column generation technique (Umetani *et al.*, 2006). We present the results obtained using the ILS method with a lower limit equal to $1.01\, f_{opt}$ ($ILS_{1.01}$), with no upper limit (ILS_{inf}).

The ILS method without an upper limit on the objects processed (ILS_{inf}) provided better setup averages in all classes (in bold type in Table 3).

Table 4 shows the average number of processed objects for the methods in each class. The Kombi method provided better averages for the objects processed in 17 classes, and the ILS method (with an upper limit equal to $1.01 f_{opt}$) obtained a better average in class 14.

Table 3 – Setup averages for each method.

Class	SHP	KOMBI	ILS1.01	ILSInf	NANLCP	HH	Symbio1	Symbio5	Symbio10
1	3,95	3,40	2,43	**1,67**	3,01	3,31	3,09	2,02	1,85
2	5,94	7,81	4,57	**1,67**	4,76	6,95	6,11	5,28	4,68
3	5,00	5,89	4,42	**2,57**	4,91	4,96	5,74	4,8	4,47
4	7,31	14,26	7,36	**2,57**	7,16	10,32	10,59	9,86	9,36
5	6,87	10,75	9,32	**4,28**	7,04	7,63	9,89	8,44	8,23
6	10,81	25,44	12,48	**4,28**	10,84	13,31	30,07	14,16	14,08
7	8,84	7,90	5,92	**5,01**	5,31	7,66	6,36	5,48	5,21
8	9,76	9,96	6,42	**5,01**	6,97	9,62	8,51	8,16	7,76
9	17,19	15,03	11,38	**9,27**	10,92	13,64	10,94	10,47	10,16
10	19,37	19,28	12,11	**9,27**	12,80	18,21	16,7	16,28	16,04
11	32,20	28,74	22,08	**16,95**	21,12	24,60	23,04	22,24	21,56
12	37,25	37,31	22,66	**16,95**	25,25	33,23	32,93	31,74	31,52
13	9,38	8,97	7,01	**6,26**	6,31	8,93	7,28	6,7	6,68
14	9,85	10,32	7,43	**6,26**	7,89	10,51	8,62	8,3	8,21
15	18,03	16,88	13,26	**11,76**	11,13	16,28	13,66	13,18	12,99
16	19,63	19,91	13,80	**11,76**	14,44	19,89	16,68	16,57	16,34
17	34,39	31,46	24,49	**21,50**	21,96	29,76	27,22	25,82	25,62
18	38,23	38,28	25,39	**21,50**	26,03	37,90	32,7	32,03	32,18

Table 4 – Average numbers of processed objects for each method.

Class	SHP	KOMBI	ILS1.01	ILSinf	NANLCP	HH	Symbio1	Symbio5	Symbio10
1	14,17	**11,49**	12,24	15,15	14,84	11,56	12,59	14,49	14,84
2	116,47	**110,25**	111,60	149,78	119,62	110,4	115,17	116,26	117,8
3	25,29	**22,13**	23,08	28,01	24,26	22,17	25,96	27,43	28,23
4	225,33	**215,93**	218,44	278,57	223,91	215,98	235,93	236,89	239,72
5	46,89	**42,96**	43,95	55,12	45,96	42,99	57,17	60,85	61,42
6	433,59	**424,71**	429,10	546,64	433,29	424,89	472,95	518,81	518,85
7	55,84	**50,21**	50,62	54,14	53,69	51,69	51,58	53,04	53,75
8	515,76	**499,52**	500,77	541,50	488,85	502,23	510,5	512,92	514,65
9	108,54	**93,67**	94,40	101,21	105,65	99,49	99,36	99,41	100,1
10	1001,59	**932,32**	936,18	1008,05	932,67	948,41	970,57	975,56	975,72
11	202,80	**176,97**	178,34	193,17	216,67	195,67	198,26	201,43	201,44
12	1873,05	**1766,20**	1773,74	1920,39	1839,63	1847,42	1932,16	1932,79	1917,4
13	69,97	**63,27**	63,53	67,61	66,77	64,20	65,23	66,51	67,32
14	643,55	632,12	**630,50**	675,50	639,88	633,26	646,77	646,31	650,52
15	136,03	**119,43**	120,53	125,86	123,93	123,90	127,39	128,43	128,87
16	1253,55	**1191,80**	1196,57	1256,92	1169,07	1197,66	1254,69	1253,79	1251,9
17	256,01	**224,68**	226,62	239,64	262,07	244,02	244,49	248,67	247,62
18	2381,54	**2242,40**	2255,12	2391,53	2247,11	2268,30	2414,07	2419,34	2422,41

Table 5 compares the total Symbio-method costs (c_1= c_2=1) to the total SHP, Kombi234, $ILS_{1.01}$, ILS_{inf}, HH and NANLCP costs. In order to compute the percentage of deviation of the total Symbio1 cost in relation to the NANLCP cost, we used the following formula:

$$V = \frac{TotalCost_{Symbio} - TotalCost_{NANLCP}}{TotalCost_{Symbio}} \times 100$$

The same thing was done when we compared Symbio5 and Symbio10 to the other methods. Compared to NANLCP, our method provides a better solution when the percentage of deviation is negative. Tables 6 and 7 show the same comparisons made in Table 5, using, however, Symbio5 (c_1=1 and c_2=5) and Symbio10 (c_1=1 and c_2=10) instead of Symbio 1.

As one can see, when c_1= c_2=1, Symbio was better than SHP in 11 classes; better than Kombi in only one class; and better than ILS_{inf}, NANLCP and HH in 10, 7 and 4 classes, respectively. When compared to $ILS_{1.01}$, Symbio1 produced inferior averages in all classes.

The best results were obtained when comparing Symbio to the other methods, using c_1=1 and c_2=5. The results we obtained were better than those of SHP in 12 classes; better than Kombi234 in 10 classes; better than ILS_{inf} in 3 classes; better than NANLCP in 4 classes; and better than HH in 8 classes. Additionally, when compared to Symbio, the ILS method (with an upper limit equal to $1.01\, f_{opt}$) obtained better total-cost averages in all classes.

Table 5 – Total-cost deviation percentage ($c_1 = c_2$) of Symbio1 in relation to SHP, Kombi, ILS, NANLCP and HH.

Class	SHP	Kombi	$ILS_{1.01}$	ILS_{inf}	NANLCP	HH
1	**-13,47**	5,31	6,88	**-6,78**	**-12,16**	5,45
2	**-0,92**	2,73	4,40	**-19,92**	**-2,49**	3,35
3	4,66	13,13	15,27	3,66	8,67	16,84
4	5,97	7,09	9,18	**-12,31**	6,69	8,94
5	24,74	24,86	25,89	12,90	26,53	32,48
6	13,19	11,74	13,91	**-8,69**	13,26	14,79
7	**-10,42**	**-0,29**	2,48	**-2,05**	**-1,80**	**-2,38**
8	**-1,24**	1,87	2,33	**-5,03**	4,68	1,40
9	**-12,27**	1,47	4,27	**-0,16**	**-5,38**	**-2,50**
10	**-3,30**	3,75	4,11	**-2,95**	4,42	2,14
11	**-5,83**	7,58	10,42	5,32	**-6,93**	0,47
12	2,87	8,96	9,39	1,43	5,37	4,49
13	**-8,62**	0,37	2,79	**-1,84**	**-0,78**	**-0,85**
14	0,30	2,02	2,74	**-3,87**	1,18	1,80
15	**-8,44**	3,48	5,43	2,49	4,44	0,62
16	**-0,14**	4,92	5,04	0,21	7,42	4,42
17	**-6,44**	6,08	8,20	4,05	**-4,34**	**-0,76**
18	1,12	7,28	7,29	1,40	7,64	6,10

Table 6 – Total-cost deviation percentage ($c_1=1$ and $c_2=5$) of Symbio5 in relation to SHP, Kombi, ILS, NANLCP and HH.

Class	SHP	Kombi	$ILS_{1.01}$	ILS_{inf}	NANLCP	HH
1	**-27,51**	**-13,69**	0,82	4,64	**-17,73**	**-12,52**
2	**-2,40**	**-4,45**	6,11	-9,78	**-0,53**	**-1,72**
3	2,27	**-0,29**	13,83	25,87	5,37	9,50
4	9,28	**-0,36**	12,13	**-1,79**	10,20	6,95
5	26,85	6,56	13,80	34,67	26,97	27,00
6	20,91	6,83	19,96	3,80	20,95	19,98
7	**-19,59**	**-10,33**	0,27	1,58	0,25	**-10,61**
8	**-1,92**	0,80	3,91	**-2,26**	5,73	0,62
9	**-21,97**	**-10,11**	0,30	2,85	**-5,30**	**-9,50**
10	**-3,78**	2,75	6,04	0,24	6,05	1,68
11	**-14,07**	**-2,51**	8,27	12,49	**-2,99**	**-1,90**
12	1,56	7,10	10,83	4,31	6,39	3,87
13	**-14,43**	**-7,50**	1,45	1,11	1,72	**-8,12**
14	**-0,72**	0,60	3,02	**-2,69**	1,25	0,29
15	**-14,08**	**-4,66**	4,01	5,24	8,21	**-5,34**
16	**-1,11**	3,51	5,62	1,59	7,68	3,05
17	**-11,73**	**-1,10**	8,22	8,82	1,59	**-3,83**
18	0,26	5,99	8,29	3,22	8,51	4,95

Symbio10 performed best using $c_1=1$ and $c_2=10$; this algorithm performed better than: SHP in 13 classes; Kombi234, in 14 classes; $ILS_{1,01}$, in 3 classes; ILS_{inf}, in 2 classes; NANLCP, in 6 classes; and HH, in 8 classes.

Table 7 – Total-cost deviation percentage (with $c_1=1$ and $c_2=10$) of Symbio10 in relation to SHP, Kombi, ILS, NANLCP and HH.

Class	SHP	Kombi	$ILS_{1.01}$	ILS_{inf}	NANLCP	HH
1	**-37,88**	**-26,71**	**-8,76**	4,68	**-25,81**	**-25,35**
2	**-6,41**	**-12,61**	4,64	**-1,13**	**-1,57**	**-8,50**
3	**-3,13**	**-10,00**	8,40	35,78	**-0,59**	1,62
4	11,69	**-7,03**	14,14	9,55	12,79	4,43
5	24,34	**-4,48**	4,79	46,77	23,51	20,48
6	21,78	**-2,87**	19,09	11,91	21,78	18,22
7	**-26,62**	**-18,08**	**-3,62**	1,54	**-0,88**	**-17,49**
8	**-3,44**	**-1,15**	4,83	0,11	6,03	**-1,03**
9	**-28,08**	**-17,33**	**-3,12**	4,02	**-6,12**	**-14,49**
10	**-4,95**	0,98	7,46	3,21	7,11	0,50
11	**-20,53**	**-10,19**	4,48	14,99	**-2,53**	**-5,58**
12	**-0,58**	4,36	11,61	6,83	6,71	2,43
13	**-18,10**	**-12,32**	0,37	3,00	3,27	**-12,63**
14	**-1,27**	**-0,37**	3,95	**-0,74**	1,93	**-0,78**
15	**-18,20**	**-10,22**	2,23	6,29	10,01	**-9,74**
16	**-2,38**	1,75	6,05	2,97	7,75	1,34
17	**-16,02**	**-6,58**	6,85	10,82	4,60	-6,98
18	**-0,71**	4,53	9,37	5,28	9,44	3,66

Table 8 – Average processing time for each method (KOMBI, HH and ILS were not implemented and tested in the same computational environment).

Class	SHP T(s)	Kombi234 T(s)	ILS T(s)	NANLCP T(s)	HH T(s)	Symbio05 T(s)
1	0.01	0.14	0.10	0.83	0.23	18.54
2	0.08	1.14	0.22	1.21	0.48	37.88
3	0.17	1.74	0.72	0.94	0.12	33.25
4	0.21	16.00	2.69	1.22	2.75	68.11
5	0.27	38.03	7.55	0.89	3.43	58.29
6	0.31	379.17	23.18	1.02	7,81	158.04
7	0.01	0.07	0.21	13.44	0.11	19.62
8	0.02	0.20	0.27	16.51	0.60	48.48
9	0.04	3.37	1.96	75.81	0.49	38.75
10	0.06	3.25	2.19	142.01	3.36	127.25
11	0.22	36.26	19.16	168.67	7.17	117.85
12	0.32	76.31	23.87	420.53	44.62	426.08
13	0.01	0.08	0.26	5.12	0.13	17.66
14	0.02	0.13	0.31	4.44	0.25	31.19
15	0.03	1.81	2.01	61.68	0.97	41.12
16	0.04	2.60	2.21	78.34	2.46	133.90
17	0.16	50.93	22.01	250.04	15.46	153.45
18	0.24	70.94	26.84	390.75	50.61	388.89

7. Conclusions and Perspectives

Upon comparing the results, we observed that on the average the Symbio method perform better with higher setup costs. Compared to the Kombi234, SHP, ILS and HH methods, one Symbio advantage is its capacity to process costs c_1 and c_2 directly within the objective function. NANCLP is the only method described in the literature that works with these costs in the objective function. However, NANLCP uses c_2 as a penalty setup-parameter, avoiding real setup costs. In fact, Moretti & Salles Neto (2008) assign c_2 a value of 100 or 300. In the real world, these costs vary depending upon several factors such as demand, delivery date, and labor costs; they can only be defined if all the data is available.

Parameters such as the mutation rate, population size and gene quantity greatly influence the final solution. For example, for problems involving items smaller than half the size of the master roll, the size of the genes was chosen in a different way in order to improve the solutions. Analyzing the results, one can see that our algorithm functions better for problems involving an average demand of around 10 items.

In Table 8, one notices that the percentage of difference between the faster and slower classes is low for Symbio, which tells us that this method is reasonably stable in relation to computing time. However, the computing time is high in comparison to the ILS method, which produces better results for most of the classes.

Finally, since this is the first study to use a genetic symbiotic algorithm in a cutting stock problem involving setup costs, we believe that improvements can be made that will result in even better methods.

Acknowledgements

We would like to thank two anonymous referees for very helpful comments on this paper. The authors have been partially supported by M.E.C. (Spain), Project MTM2007-063432. The second author is also supported by CNPq (Brazil), Project 307907/2007-4.

References

(1) Allaby, M. (1998). *Dictionary of Ecology*. Oxford University Press, 1998, New York.

(2) Boleta, D.A.F.; Araújo, S.A.; Constantino, A.A. & Poldi, K.C. (2005). Uma heurística para o problema de corte e estoque unidimensional inteiro. *Anais do XXXVII Simpósio Brasileiro de Pesquisa Operacional*, 1869-1879.

(3) Bremermann, H.J. (1962). Optimization through evolution and recombination. **In**: *Self-Organizing Systems* [edited by M.C. Yovits, G.T. Jacobi and G.D. Goldstine], 93-106, Spartan Books.

(4) Chang, M.; Ohkura, K.; Ueda, K. & Sugiyama, M. (2002). A symbiotic evolutionary for dynamic facility layout problem. **In**: *Proceedings of the Evolutionary Computation*, 1745-1750.

(5) Diegel, A. (1988). Cutting paper in Richards Bay: dynamic local and global optimization in the trim problem. *Orion*, **3**, 42-55.

(6) Diegel, A.; Chetty, M.; Van Schalkwyck, S. & Naidoo, S. (1993). Setup combining in the trim loss problem – 3-to-2 & 2-to-1. Working paper, Business Administration, University of Natal, Durban, First Draft.

(7) Dowsland, K. & Dowsland, W. (1992). Packing Problems. *European Journal of Operational Research*, **56**, 2-14.

(8) Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, **56**, 145-159.

(9) Eguchi, T.; Hirasawa, K. & Hu, J. (2003). Symbiotic Evolutional Models in Multiagent Systems. *The 2003 Congress on Evolutionary Computation*, **2**, 739-746.

(10) Eilon, S. (1960). Optimizing the shearing of steel bars. *Journal of Mechanical Engineering Science*, **2**, 129-142.

(11) Foester, H. & Wäscher, G. (2000). Pattern Reduction in One-dimensional Cutting-Stock Problem. *International Journal of Prod. Res.*, **38**, 1657-1676.

(12) Fogel, D. (1995). *Evolutionary computation: toward a new philosophy of machine intelligence*. IEEE Press, New York.

(13) Fraser, A.S. (1957). Simulation of genetic systems by automatic digital computers: I. Introduction. *Austral. J. Biol. Sci.*, **10**, 484-491.

(14) Gau, T. & Wäscher, G. (1995). CUTGEN1: A Problem Generator for the Standard One-dimensional Cutting Stock Problem. *European Journal of Operational Research*, **84**, 572-579.

(15) Gilmore, P.C. & Gomory, R.E. (1961). A Linear Programming Approach to the Cutting Stock Problem. *Operations Research*, **9**, 849-859.

(16) Gilmore, P.C. & Gomory, R.E. (1963). A Linear Programming Approach to the Cutting Stock Problem. *Operations Research*, **11**, 864-888.

(17) Haessler, R. (1975). Controlling Cutting Pattern Changes in One-Dimensional Trim Problems. *Operations Research*, **23**, 483-493.

(18) Hirasawa, K.; Ishikawa, I.; Hu, J.; Jin, C. & Murata, J. (2000). Genetic Symbiosis Algorithm. *Proceedings of the Congress on Evolutionary Computation*, **2**, 02-xxvi.

(19) Holland, J.H. (1962). Outline for a logical theory of adaptive systems. *J. Assoc. Comput. Mach.*, **3**, 297-314.

(20) Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

(21) Khalifa, Y.; Salem, O. & Shahin, A. (2006). Cutting Stock Waste Reduction Using Genetic Algorithms. *Proceedings of the 8th Conference on Genetic and evolutionary computation*, 1675-1680.

(22) Kantorovich, L.V. (1960). Mathematical Methods of Organizing and Planning Production. *Management Science*, **6**, 366-422.

(23) Kim, Y.K.; Kim, J.Y. & Kim, Y. (2000). A coevolutionary algorithm for balancing and sequencing in mixed model assembly lines. *Applied Intelligence*, **13**, 247-258.

(24) Kim, J.Y.; Kim, Y. & Kim, Y.K. (2001). An endosymbiotic evolutionary algorithm for optimization. *Applied Intelligence*, **15**, 117-130.

(25) Kim, Y.K.; Kim, J.Y. & Kim, Y. (2006). An endosymbiotic evolutionary algorithm for the integration of balancing and sequencing in mixed-model U-lines. *European Journal of Operational Research*, **168**, 838-852.

(26) Liang, K.; Yao, X.; Newton, C. & Hoffman, D. (2002). Evaluation of algorithms for one-dimensional cutting. *Computers & Operations Research*, **29**, 1207-1220.

(27) Mao, J.; Hirasawa, K.; Hu, J. & Murata, J. (2000). Genetic Symbiosis Algorithm for Multiobjective Optimization Problem. *Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication*, 137-142.

(28) Margulis, L. (1981). *Symbiosis in Cell Evolution*. W.H. Freeman, San Francisco.

(29) Metzger, R.W. (1958). *Stock Slitting, Elementary Mathematical Programming*. Wiley.

(30) Moretti, A.C. & Salles Neto, L.L. (2008). Nonlinear cutting stock problem model to minimize the number of different patterns and objects. *Computational & Applied Mathematics*, **27**, 61-78, 2008.

(31) Paull, A.E. & Walter, J.R. (1954). The trim problem: an application of linear programming to the manufacture of news-print paper. *Presented at Annual Meeting of Econometric Society*, Montreal, 10-13.

(32) Pianka, E.R. (1994). *Evolutionary Ecology*. HarperCollins College Publisher, New York.

(33) Pierce, J.F. (1964). *Some Large-Scale Production Scheduling Problems in the Paper Industry*. Englewood Cliffs, Prentice Hall.

(34) Potter, M.A. (1997). The design and analysis of a computational model of cooperative coevolution. Ph.D. Dissertation, George Mason University.

(35) Stadler, H. (1990). A one-dimensional cutting stock problem in the aluminium industry and its solution. *European Journal of Operational Research*, **44**, 209-223.

(36) Umetani, S.; Yagiura, M. & Ibaraki, T. (2003). One Dimensional Cutting Stock Problem to Minimize the Number of Different Patterns. *European Journal of Operational Research*, **146**, 388-402.

(37) Umetani, S.; Yagiura, M. & Ibaraki, T. (2006). One Dimensional Cutting Stock Problem with a Given Number of Setups: A Hybrid Approach of Metaheuristics and Linear Programming. *Journal of Mathematical Modelling and Algorithms*, **5**, 43-64.

(38) Tsujimura, Y.; Mafune, Y. & Mitsuo, G. (2001). Effects of symbiotic evolution in genetic algorithms for job-shop scheduling. *IEEE*.

(39) Vanderbeck, F. (1999). Computational study of a column generation algorithm for bin packing and cutting stock problems. *Math. Program.*, **86**, 565-594.

(40) Vanderbeck, F. (2000). Exact Algorithm for Minimising the Number of Setups in the One-Dimensional Cutting Stock Problem. *Operations Research*, **48**, 915-926.

(41) Von Zuben, F.J. (2003). Computação Evolutiva: uma abordagem pragmática. UNICAMP, Available on: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/tutorial/tutorialEC.pdf>.

(42) Wäscher, G.; Carow, P. & Muller, H. (1985). Entwicklung eines flexiben Verfahrens für Zuschneideprobleme in einem Kaltwalzwerk. *Zeitschrif für Operations Research*, **29**, B209-B230.

(43) Wäscher, G. (1990). An LP-based approach to cutting stock problems with multiple objectives. *European Journal of Operational Research*, **44**, 175-184.

(44) Wäscher, G. & Gau, T. (1996). Heuristics for the Integer One-dimensional Cutting Stock Problem: a computational study. *Operations Research Spektrum*, **18**, 131-144.

(45) Watson, R.A. & Pollack, J.B. (1999). How Symbiosis Can Guide Evolution. *Advances in Artificial Life: 5th European Conference*, Springer.

(46) Yanasse, H.H. & Limeira, M. (2006). A hybrid heuristic to reduce the number of different patterns in cutting stock problems. *Computer & Operations Research*, **33**, 2744-2756.