A Genetic Algorithm Approach to Solve the Synchronized and Integrated Two-Level Lot Sizing and Scheduling Problem

Claudio Fabiano Motta Toledo
Departamento de Ciência da Computação
Universidade Federal de Lavras
C.P. 3037, 37200-000, Lavras, MG, Brasil
claudio@dcc.ufla.br

Paulo Morelato França
Fac. de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas
Rua Albert Einstein, 400, C.P. 6101, 13083-852, Campinas, SP, Brazil
franca@densis.fee.unicamp.br

Reinaldo Morabito
Departamento de Engenharia de Produção
Universidade Federal de São Carlos
Via Washington Luiz, km. 235, C.P. 676, 13565-905, São Carlos, SP, Brazil
morabito@power.ufscar.br

Alf Kimms
Chair of Logistic and Traffic Management
Dept. of Technology and Operations Mgmt
University of DuisburgEssen
47048, Duisburg, Germany
alf.kimms@uniduisburgessen.de

ABSTRACT

This paper proposes a Genetic Algorithm approach to solve the Synchronized and Integrated Two-Level Lot Sizing and Scheduling Problem (SITLSP). This problem can be found in some industrial settings, mainly soft drink companies, where the production process involves two interdependent levels with decisions concerning raw material storage and soft drink bottling. The challenge is to simultaneously determine the lot sizing and scheduling of raw materials in tanks and soft drinks in the bottling lines, where setup costs and times depend on the previous items stored and bottled. A Multi-Population Genetic Algorithm approach with a particular representation of solutions for individuals and a hierarchical ternary tree structure for populations is proposed. The computational study reported here reveals that this procedure is the only viable method to solve real-world and industrial instances, and it gives better results on the medium problems than an exact approach.

1. Introduction

The production problem studied in this article was motivated by a real situation found in some industrial settings, such as soft drink companies. This production process usually has two interdependent levels. In the first level, decisions about the amount and the time the raw materials have to be stored in each one of the available tanks must be made. Similarly, in the second level, the lot size of each demanded item and its corresponding scheduling in each line must be also determined. A lot sizing and scheduling problem has to be solved in each one of these two-levels (Figure 1).

The raw material is the flavor of the soft drink that is bottled on a production line, which comes from storage tanks with a limited storage capacity. For technical reasons, a tank is only filled up when empty and two different raw materials cannot be stored at the same time in the same tank. A sequence-dependent setup time occurs to clean and fill up a tank, even if the same soft drink is replaced. Nothing can be pumped to a production line from the tank during the setup time. One tank can be connected to several production lines which will share the same raw material. Also, one production line can be connected with several tanks storing the same raw material. The final product (item) is defined by the flavor of the soft drink and the type of container (glass bottles, plastic bottles or cans) of different sizes. Several products can share a common production line and several lines can produce the same product in parallel. There is a sequence dependent setup time whenever a line has two different products switched. In general, the weekly demands have to be met within a time horizon of four weeks.

A synchronization problem occurs because the production in lines and the storage in tanks must be compatible with each other throughout the time horizon. The excessive number of final products leads to inventory costs. There are also inventory costs for the storage of raw material in tanks during several time periods. There is a production cost for each unit of product and raw material. The sequence-dependent setup costs are proportional to the sequence-dependent setup times in lines and tanks, respectively. The challenging aspect of the problem addressed is the combination of all these issues in an interdependent two-level problem. For these reasons, this problem is called the Synchronized and Integrated Two-level Lot Sizing and Scheduling Problem (SITLSP).

The capacitated lot sizing and scheduling problem is a NP-hard optimization problem. If setup times are present, the problem of finding a single feasible solution becomes NP-complete already. Discussion and reviews on this problem issues can be found in Drexl and Kimms (1997) and Karimi et al. (2003). Approaches to multi-level problems are presented by Kimms (1997) and Berreta et al. (2005). These problems with parallel machines are also studied in Meyr (2002). A mathematical model for the SITLSP is proposed in Kimms et al. (2005) and Toledo (2005), where several instances were solved using the GAMS/Cplex package (2000).

The method proposed in this paper to solve the SITLSP is a Multi-Population Genetic Algorithm with a hierarchical ternary tree structure. The Genetic Algorithm (GA) is a search procedure inspired by biological evolution process, see Holland (1975). Lot-sizing and scheduling problems with sequence-dependent setup times are solved using GA in Sikora (1996).
and Özdamar and Birbil (1998). An elaborate representation of a solution such as individuals for lot-sizing and scheduling problems can be found in Dorndorf and Pesch (1995) and Dellaert et al. (2000). All the mentioned works have specific genetic operator designed to deal with solutions representation (individuals) for lot size and scheduling problems. Mendes (2003) reported better results solving different optimization problems using a multi-population Genetic Algorithm with a hierarchical ternary tree structure instead of a non-structured single population GA. In addition, Franca et al. (2001) found better results solving the total tardiness single machine scheduling problem using GA with hierarchically structured populations.

The GA proposed in this work is described in sections 2. The computational results are summarized next for small, large and industrial sized instances. Concluding remarks are presented in the last section.

2. Genetic Algorithm

The GA was implemented using the NP-Opt framework proposed by Mendes (2003). This framework is based on evolutionary computation techniques to address NP-hard problems. Figure 2 has the NP-Opt pseudo-code for the multi-population GA.

```plaintext
Method MultiPopulationGeneticAlg
begin
    repeat
        for i = 1 to numberOfPopulations do
            initializePopulation(pop( i ));
            evaluatePopulationFitness(pop( i ));
            structurePopulation(pop( i ));
            repeat
                for j = 1 to numberOfCrossover do
                    selectParents(individualA, individualB);
                    newInd = crossover(individualA, individualB);
                    if (executeMutation newInd) then
                        newInd = mutation(newInd);
                    evaluateFitnessIndividual(newInd);
                    insertPopulation(newInd, pop(i ));
                end
                structurePopulation(pop(i ));
            until(populationConvergence pop(i ));
        end
        for i = 1 to numberOfPopulations do
            executeMigration pop(i );
        end
    until(stop criterion)
end
```

Figure 2: Pseudo-code for the Multi-Population Genetic Algorithm.

The algorithm executes a fixed number of crossovers in each population while there is no convergence. A mutation can happen over the new individual created by the crossover operator. The population convergence occurs when there are no new individuals inserted in any population. At this point, a migration among populations is executed since the stop criterion has not been satisfied yet. A new initialization of the populations will occur, but the best individual and the migrated individuals are kept. Details on how this pseudo-code was implemented are described next.

2.1 Individual and Genetic Operators

A new representation of a solution as an individual is proposed in Figure 3. This representation is close to the one presented by Kimms (1999) that uses a two-dimensional matrix with assignment rules for a multi-level proportional lot sizing and scheduling problem with multiple machines.
Each individual is a two-dimensional matrix with T rows and N columns. The rows represent the time periods \( t_1, t_2, ..., t_T \) in which the time horizon T is divided. The columns represent the genes and there can be a different number of genes per period. Each gene is an entry \( (m,n) \), \( m \in T \), \( n \in N \), in this matrix with the following data:

- \( P_{mn} \): product in gene \( n \) to be produced in macro-period \( m \).
- \( D_{mn} \): lot size of product \( P_{mn} \).
- \( SL_{mn} \): sequence of lines where \( D_{mn} \) can be produced.
- \( STK_{mn} \): sequence of tanks where the raw material of \( D_{mn} \) can be stored.

The demand \( d_i \) of product \( i \) in period \( t \) is divided into many lots \( (D_{mn}) \) and randomly distributed among the genes in periods \( t \), \( t-1 \), \( t-2 \), ..., 1. The sequences \( SL_{mn} \) and \( STK_{mn} \) are randomly generated with length \( k \). Sequence \( SL_{mn}=(\alpha_1, ..., \alpha_k) \) with \( \alpha_i \) is a possible line number and \( L \) is the number of lines. The \( \alpha_i \) is taken from \( L \) possible values.

Sequence \( STK_{mn}=(\beta_1, ..., \beta_k) \) with \( \beta_i \in \{1,2,...,2L\} \), where \( \beta_i \) tells where and how the raw material will be stored. The parameter \( L \) is the number of tanks. The \( \beta_i \) is taken from \( 2L \) possible values, but the real tank number \( j \) is obtained from \( \beta_i \) doing:

\[
\begin{align*}
    j = \begin{cases} 
        \beta_i, & 1 \leq \beta_i \leq L \\
        \beta_i - L, & L < \beta_i \leq 2L 
    \end{cases}
\end{align*}
\]

If \( 1 \leq \beta_i \leq L \), the tank \( j=\beta_i \) will be occupied after the raw material previously stored has been used. This forces the method to find solutions where there is a partial use of the tank capacity. If \( L < \beta_i \leq 2L \), the tank \( j=\beta_i - L \) will be immediately occupied. This forces the method to find solutions where the tank capacity is completely used. These conditions have some exceptions. If tank \( j \), selected by one of the previous criterions, stores a raw material different from the raw material of the product \( P_{mn} \), it will be necessarily occupied after the raw material previously stored has been used. The same will happen if tank \( j \) is completely full. On the other hand, if tank \( j \) is empty, it will be immediately occupied. If \( j \) is not empty, but the raw material stored is the same of \( P_{mn} \) and the minimum tank capacity have not been satisfied, this tank will be also occupied immediately.

An example clarifies the solution representation as an individual. Suppose two products (P1 and P2) where each product has a demand of 100 units to be filled in period \( t_1 \) and another 100 units to be filled in period \( t_2 \). The products have different raw materials. There are also two lines and two tanks available to produce and store all these raw materials and products, respectively. Figure 4 gives us two possible individuals. The demands are distributed in their respective periods in individual 1, but the demand of P2 in \( t_1 \) is split between two genes. In individual 2, part of the P1 demand in \( t_2 \) is produced in period \( t_1 \). Note that sequence of lines (\( SL_{mn} \)) and tanks (\( STK_{mn} \)) can repeat values of \( \alpha_i \in \{1,2\} \) and \( \beta_i \in \{1,2,3,4\} \) for \( L=L=2 \) and \( k=4 \) (length).
Once defined a representation of the solution as an individual, the next step is to determine genetic operators. Previous computational experiments with several crossover operators, see Toledo (2005), revealed that the best result was obtained with the uniform crossover. Individuals 1 and 2 (figure 4) are used in figure 5 to show how the uniform crossover operator works.

Sequences $SL_{mn}$ and $ST_{mn}$ are not shown in figure 5 because the new individual (Child) will inherit them without changes. For each gene in the same position in Ind1 and Ind2, a random value $\lambda \in [0,1]$ is generated. If $\lambda < 0.5$, Child inherits the gene of Ind1; otherwise, Child inherits the gene of Ind2. The genes selected are shaded in figure 5. As there are more genes in one period of Ind2 than in the same period of Ind1, Child continues inheriting those genes from Ind2 when $\lambda \geq 0.5$. It is not allowed excessive demand in Child. For this reason, the gene circled in Ind1 is not inherited. In this case, the gene in the same position of Ind2 must be inherited, since the same problem does not happen again. If a lack of demand occurs at the end, a repair procedure will take place and the demand deficit in some period is inserted in it.

The mutation operator aims to keep diversity in a population avoiding premature convergence (Figure 6). Three types of mutations were developed and they basically swap gene positions. The first mutation type swaps the positions of two randomly selected genes in the same period. In the second type, the selected gene is removed and inserted into another position, which is also randomly selected. The third type swaps the positions of two chosen genes that are in different periods. A gene swap will not take place, if it can violate the demand satisfaction. The mutation type is randomly chosen.

![Figure 4: Two individuals](image)

Once defined a representation of the solution as an individual, the next step is to determine genetic operators. Previous computational experiments with several crossover operators, see Toledo (2005), revealed that the best result was obtained with the uniform crossover. Individuals 1 and 2 (figure 4) are used in figure 5 to show how the uniform crossover operator works.

Sequences $SL_{mn}$ and $ST_{mn}$ are not shown in figure 5 because the new individual (Child) will inherit them without changes. For each gene in the same position in Ind1 and Ind2, a random value $\lambda \in [0,1]$ is generated. If $\lambda < 0.5$, Child inherits the gene of Ind1; otherwise, Child inherits the gene of Ind2. The genes selected are shaded in figure 5. As there are more genes in one period of Ind2 than in the same period of Ind1, Child continues inheriting those genes from Ind2 when $\lambda \geq 0.5$. It is not allowed excessive demand in Child. For this reason, the gene circled in Ind1 is not inherited. In this case, the gene in the same position of Ind2 must be inherited, since the same problem does not happen again. If a lack of demand occurs at the end, a repair procedure will take place and the demand deficit in some period is inserted in it.

The mutation operator aims to keep diversity in a population avoiding premature convergence (Figure 6). Three types of mutations were developed and they basically swap gene positions. The first mutation type swaps the positions of two randomly selected genes in the same period. In the second type, the selected gene is removed and inserted into another position, which is also randomly selected. The third type swaps the positions of two chosen genes that are in different periods. A gene swap will not take place, if it can violate the demand satisfaction. The mutation type is randomly chosen.

![Figure 5: Uniform crossover](image)

Sequences $SL_{mn}$ and $ST_{mn}$ are not shown in figure 5 because the new individual (Child) will inherit them without changes. For each gene in the same position in Ind1 and Ind2, a random value $\lambda \in [0,1]$ is generated. If $\lambda < 0.5$, Child inherits the gene of Ind1; otherwise, Child inherits the gene of Ind2. The genes selected are shaded in figure 5. As there are more genes in one period of Ind2 than in the same period of Ind1, Child continues inheriting those genes from Ind2 when $\lambda \geq 0.5$. It is not allowed excessive demand in Child. For this reason, the gene circled in Ind1 is not inherited. In this case, the gene in the same position of Ind2 must be inherited, since the same problem does not happen again. If a lack of demand occurs at the end, a repair procedure will take place and the demand deficit in some period is inserted in it.

The mutation operator aims to keep diversity in a population avoiding premature convergence (Figure 6). Three types of mutations were developed and they basically swap gene positions. The first mutation type swaps the positions of two randomly selected genes in the same period. In the second type, the selected gene is removed and inserted into another position, which is also randomly selected. The third type swaps the positions of two chosen genes that are in different periods. A gene swap will not take place, if it can violate the demand satisfaction. The mutation type is randomly chosen.

![Figure 6: Mutation types](image)

2.2 Fitness Value

The fitness value of one individual is determined by a decoding procedure, which takes
the information encoded in each gene and translates it into a solution of the problem (figure 7).

![Figure 7: Decoding procedure](image)

A pair \((\alpha_i, \beta_i)\) is selected from the sequence of values in \(SL_{mn}\) and \(ST_{kmn}\) while \(D_{mn} > 0\). If \((\alpha_i, \beta_i)\) returns a line and a tank with available capacity, \(D_{mn}\) or a fraction of \(D_{mn}\) is scheduled. If the capacity is not enough to produce \(D_{mn}\) or it was enough to produce only a fraction of \(D_{mn}\), the next pair \((\alpha_{i+1}, \beta_{i+1})\) is selected. These steps start with the first gene in the last period of the individual. Thus, they are repeated over each gene until reach the last gene in the first period. This backward process aims to postpone setups and processing time of products and raw materials in lines and tanks. However, there is no guarantee that all demands will be produced at the end of the decoding.

The complete decoding of one individual returns a lot size and schedule for lines and tanks. This schedule is a possible solution for the problem that is evaluated next using the fitness function. The objective function of the mathematical formulation described in Kimms et al. (2005) and Toledo (2005) corresponds to the fitness function used by the GA. To sum up, this value is determined adding up all costs that happen in the final schedule: setup, production and inventory costs for products and raw materials in lines and tanks, respectively. If some demands are not satisfied, a high penalty cost per unit is also added to the fitness. The best individual is the one whose decoding process determines a final schedule with a minimum cost value.

### 2.3 Population Structure and Migration

A multi-population GA can allow a more effective exploration in the solution space because populations that evolve separately will present different characteristics second the genetic drift idea, see Weiner (1995). The Multi-Population proposed is constituted by a hierarchical two-level ternary tree structure. Each structure has 4 clusters of 4 individuals each that are arranged in two levels (1 cluster in level 1 and 3 clusters in level 2). Each cluster has a node leader (best individual) and 3 other nodes called supporters (figure 8).

![Figure 8: Hierarchical structure](image)

The crossover describe previously is carried out over a cluster (selected at random) and always selects a supporter node (also randomly chosen) and its respective leader node. If better, the new individual (Child) will replace the parent with the worst fitness value. Otherwise, the new individual is not inserted into this population. After the crossover, adjustments are done in the tree to keep the cluster hierarchically structure (the best is the leader). The populations converge when no new individuals are inserted after a fixed number of crossover has been completed (see figure 2). At this point, a migration is executed where a copy of each best individual is inserted into the next population. This copy will replace some individual randomly selected, except the best one of each population. A new initialization of the population occurs after the migration process. However, the best individuals and individuals that have just migrated are kept.
3. Computational Results

Set of instances were established for the SITLSP by Toledo (2005) based on data provided by a Brazilian soft drink company. These instances were separated here in three groups: small-to-moderate size, moderate-to-large size and industrial instances.

3.1 Small-to-Moderate Size Instances

Tables 1 and 2 show parameter values used to create small-to-moderate size instances.

<table>
<thead>
<tr>
<th>Param.</th>
<th>Values</th>
<th>Meaning</th>
<th>Param.</th>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| $L$    | {2;3;4} | Number of lines          | $h_j$  | 1($/u$) | Unity inventory cost of  
	production $j$           |
| $\bar{L}$ | {2;3} | Number of tanks          | $\bar{h}_j$ | 1($/u$) | Unity inventory cost of  
	raw material $j$        |
| $J$    | {2;3;4} | Number of products       | $v_{jl}$ | 1($/u$) | Unity production cost of  
	product $j$ in line $l$ |
| $\bar{J}$ | {1;2} | Number of raw materials  | $\bar{v}_{jk}$ | 1($/u$) | Unity production cost of  
	raw material $j$ in tank $k$ |
| $T$    | {1;2;3;4} | Number of periods       | $Q_k$  | 5000| Maximum capacity of tank $k$  
	(liters)               |
| $C$    | 5     | Capacity of each period  | $Q^*$  | 1000 | Minimum capacity of tank $k$  
	(liters)               |

Table 1 – Parameter values and meaning

<table>
<thead>
<tr>
<th>Param.</th>
<th>Ranges</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$st_{ijl}$</td>
<td>U[0.5; 1]</td>
<td>Setup time (hours) from product $i$ to $j$ in line $l$</td>
</tr>
<tr>
<td>$sst_{ijk}$</td>
<td>U[1; 2]</td>
<td>Setup time (hours) from raw material $i$ to $j$ in tank $k$</td>
</tr>
<tr>
<td>$p_{ij}$</td>
<td>U[1000; 2000]</td>
<td>Process time (units/hour) of product $i$ in line $l$</td>
</tr>
<tr>
<td>$d_{jt}$</td>
<td>U[500; 10000]</td>
<td>Demand (units) of product $j$ in period $t$.</td>
</tr>
<tr>
<td>$r_{ji}$</td>
<td>U[0.3; 3]</td>
<td>Conversion factor</td>
</tr>
</tbody>
</table>

Table 2 - Parameter ranges

The conversion factor $r_{ji}$ in table 2 determines how many liters of raw material $i$ are necessary to produce one unit of product $j$. The setup costs in lines $(s_{c_{ijl}})$ and tanks $(s_{c_{ijk}})$ are computed doing $s_{c_{ijl}} = f^*s_{t_{ijl}}$ and $s_{c_{ijk}} = f^*s_{t_{ijk}}$, respectively. The most adequate value for $f$ is 1000 given that it provides a suitable trade off between the different terms of the fitness, second Toledo (2005). It was chosen 9 main parameter combinations (Comb.) per period limiting to 4x9=36 set of possible instances. In each set, 10 replications are randomly generated resulting in 4x9x10 = 360 instances (table 3).

<table>
<thead>
<tr>
<th>Comb.</th>
<th>$L/\bar{L}/J/\bar{J}$</th>
<th>Comb.</th>
<th>$L/\bar{L}/J/\bar{J}$</th>
<th>Comb.</th>
<th>$L/\bar{L}/J/\bar{J}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2/2/2/1</td>
<td>4</td>
<td>3/3/2/1</td>
<td>7</td>
<td>4/4/2/1</td>
</tr>
<tr>
<td>2</td>
<td>2/2/3/2</td>
<td>5</td>
<td>3/3/3/2</td>
<td>8</td>
<td>4/4/3/2</td>
</tr>
</tbody>
</table>

Table 3 – Set of small-to-moderate instances

First, the computational results for these instances were obtained using the modeling language GAMS IDE, version 2.0.10.0, and the solver CPLEX, version 7. The CPLEX handles mixed integer problems using a branch and cut algorithm, see GAMS (2000). The GAMS/CPLEX ran on each instance only once during the time limit of 1 hour. Two kinds of solutions were returned: the optimal solution or the best feasible solution achieved up to the time limit. Table 4 shows the number of optimal solutions found (Opt.) and the average CPU time spent by GAMS/CPLEX on each combination.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>2.08</td>
<td>10</td>
<td>28.27</td>
<td>2</td>
<td>3146.80</td>
<td>1</td>
<td>3377.44</td>
</tr>
</tbody>
</table>
Table 4 – GAMS/CPLEX results for $T=1$ and 2.

GAMS/CPLEX solves optimally most of the instances with $T=1$ (77 out of 90 instances). It begins to have problems to return optimal solution within the time limit for $T=2$. Only 25 out of 90 instances were optimally solved. This number decreases considerably for $T=3$ and 4, when only 11 out 180 instances were optimally solved. More about the mathematical model of the SITLSP and these instances resolutions using GAMS/CPLEX were reported in Kimms et al. (2005). The next step was to solve the instances using the GA. In the combinations where the GAMS/CPLEX spent up to 0.5 hour (CPU time) in average to find optimal solutions, the GA ran on during 360 seconds. This execution was repeated 5 times over each instance. In this way, the method spent 0.5 hour per instance. In the combinations where the GAMS/CPLEX took more than 0.5 hour, the GA ran on during 1200 seconds. This execution was repeated 3 times over each instance, that is, an execution time of 1 hour per instance. The GA was set with 3 populations structured in ternary trees of 13 individuals each. The crossover rate of 1.5 and the mutation rate of 0.7 were adjusted based on previous tests. The following percentage deviation was used to compare the GA and GAMS/CPLEX solutions:

$$\text{Dev}(\%) = 100 \left( \frac{Z - \bar{Z}}{Z} \right)$$

where $Z$ is the final GA solution and $\bar{Z}$ is the solution returned by GAMS/CPLEX. The average (Avg), maximum (Max) and minimum (Min) deviation values are listed in Table 5.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 5: GA deviation from Gams/CPLEX results for $T=1, 2, 3$ and 4.

The Avg considers the deviation of the final solutions of all (three or five) GA ran on each instance from the best GAMS/CPLEX solution. Max is the same average deviation, but it considers only the worse solution found by GA in its runs on each instance. Min is the average deviation considering only the best solution found by GA, after the executions in each instance. GA was able to optimally solve most of the instances with $T=1$. For $T=2, 3$ and 4, some average deviation are negative, meaning that the GA procedure found in average better solutions than the (non-optimal) solutions returned by GAMS/CPLEX. GA outperform in average the GAMS/CPLEX best feasible solutions in 12 out of 36 set of instances.
3.2 Large-to-Moderate Size

There are more products and raw materials in the large-to-moderate size instances, so a larger number of periods and a more available capacity per period are considered (Table 6).

<table>
<thead>
<tr>
<th>Param.</th>
<th>Values</th>
<th>Meaning</th>
<th>Param.</th>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>{5; 8; 4}</td>
<td>Number of lines</td>
<td>J</td>
<td>{10; 15;}</td>
<td>Number of products</td>
</tr>
<tr>
<td>( \overline{L} )</td>
<td>{5; 6}</td>
<td>Number of tanks</td>
<td>( \overline{J} )</td>
<td>{5; 8}</td>
<td>Number of raw materials</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>Capacity of each period</td>
<td>T</td>
<td>{4,8,12}</td>
<td>Number of periods</td>
</tr>
</tbody>
</table>

Table 6 – Parameter values and meaning

Table 7 has the parameter combination for large instances, where 10 replications were randomly generated for \( T=4, 8 \) and 12 resulting in \( 3 \times 4 \times 10 = 120 \) instances in total.

<table>
<thead>
<tr>
<th>Comb.</th>
<th>( L/J/\overline{J} )</th>
<th>Comb.</th>
<th>( L/J/\overline{J} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5/5/10/5</td>
<td>12</td>
<td>8/5/10/5</td>
</tr>
<tr>
<td>11</td>
<td>5/6/15/8</td>
<td>13</td>
<td>8/6/15/8</td>
</tr>
</tbody>
</table>

Table 7 – Set of large instances

All the other parameters not mentioned here kept their previous values. The GAMS/CPLEX was unable to find at least a feasible solution in all instances within 1 hour. Due to this, Table 8 presents only the model figures.

<table>
<thead>
<tr>
<th>Comb.</th>
<th>( T=4 )</th>
<th>( T=8 )</th>
<th>( T=12 )</th>
<th>( T=4 )</th>
<th>( T=8 )</th>
<th>( T=12 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin.</td>
<td>10 4810</td>
<td>71961</td>
<td>23857</td>
<td>9670</td>
<td>143961</td>
<td>47549</td>
</tr>
<tr>
<td>Cont.</td>
<td>8724</td>
<td>208172</td>
<td>65166</td>
<td>17508</td>
<td>416388</td>
<td>132258</td>
</tr>
<tr>
<td>Bin.</td>
<td>6670</td>
<td>110553</td>
<td>33778</td>
<td>13390</td>
<td>221145</td>
<td>68114</td>
</tr>
<tr>
<td>Cont.</td>
<td>12180</td>
<td>321272</td>
<td>94963</td>
<td>24420</td>
<td>642588</td>
<td>188687</td>
</tr>
<tr>
<td>Const.</td>
<td>14530</td>
<td>215961</td>
<td>70657</td>
<td>26292</td>
<td>624604</td>
<td>197318</td>
</tr>
</tbody>
</table>

Table 8 – Model size for the set of instances \( T=4, 8 \) and 12

The branch and cut algorithm could not handle the high number of variables and constraints present in the models related to each moderate-to-large size instances. Therefore, there is no GAMS/CPLEX solution to compare with GA solutions. The GA was executed 3 times over each instance with 3600 seconds assigned to each run. The multi-population algorithm used 3 populations structured again in ternary trees and has the same crossover and mutation rate as before. Table 9 shows the average CPU time (in seconds) spent by GA to find the first feasible solution without demand penalty and the best feasible solution.

<table>
<thead>
<tr>
<th>Comb.</th>
<th>( T=4 )</th>
<th>( T=8 )</th>
<th>( T=12 )</th>
<th>( T=4 )</th>
<th>( T=8 )</th>
<th>( T=12 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin.</td>
<td>10 4.69</td>
<td>5.87</td>
<td>13.48</td>
<td>1648.83</td>
<td>2083.14</td>
<td>3039.9</td>
</tr>
<tr>
<td>Cont.</td>
<td>11 5.71</td>
<td>11.41</td>
<td>18.72</td>
<td>2278.55</td>
<td>2898.03</td>
<td>3244.43</td>
</tr>
<tr>
<td>Bin.</td>
<td>12 3.98</td>
<td>8.01</td>
<td>13.5</td>
<td>2104.63</td>
<td>2839.96</td>
<td>3047.02</td>
</tr>
<tr>
<td>Cont.</td>
<td>13 6.16</td>
<td>17.66</td>
<td>22.61</td>
<td>2119.5</td>
<td>2975.2</td>
<td>3116.54</td>
</tr>
</tbody>
</table>

Table 9 - Average CPU time in seconds to find the first and the best feasible solutions

At least in the Brazilian soft drink company that provides the parameters for this computational study, solutions with demands not satisfied are useless for practical proposes. The GA took less than 1 minute to reach the first feasible solution without demand penalty. The GA average CPU time to find the best feasible is close to the time limit (3600 seconds for each run) when the number of periods increase. Thus, a larger execution time can help the method to return better results for these instances. The method was able to improve considerable the first feasible solution found returning a better value at the end in all set of instances. Figure 9 depicts the percentage of this improvement. The percentage value is determined doing Improvement(%) = - Dev(%) where \( Z \) is the value of the final solution and \( \overline{Z} \) is the value of the first feasible solution in equation (2).
Figure 8: Average improvement for combinations 10, 11, 12 and 13 of \( T = 4, 8 \) and 12.

The GAMS/CPLEX package returned optimal solutions for moderate-to-large sized instances only when all the binary variables were relaxed in the mathematical model. In this case, the exact approach was used to provide lower bounds for those instances where all the models variables are continuous (Figure 9).

Figure 9: GA solutions deviation from lower bounds.

The GAMS/CPLEX does not return a sophisticated lower bound which allows evaluating better the GA solution deviation. The gaps of the lower bounds are relatively large which prejudice a better evaluation of the GA results. These lower bounds do not give a precise idea about how far the GA solutions are from optimal solutions. In Figure 9, the deviations become large when the number of products (and raw materials) increases during the same period and for the same number of lines. In combinations 10 and 11, the deviation increases from 20.3 up to 31.3 in \( T = 4 \), from 19.7 up to 37.4 in \( T = 8 \) and from 27.2 up to 52.2 in \( T = 12 \). The same almost happened with combinations 12 and 13 where the only exception occurred in \( T = 4 \) (deviation value decreased from 24.6 to 23.8). Another parameter which has a considerable impact in the deviations from lower bounds seems to be the number of period itself. The exact approach cannot deal with the complexity of the mathematical model associated to the moderate-to-large size instances. The computational results show that the GA is the only viable alternative proposed here to solve these moderate-to-large size instances.

### 3.3 Industrial Instances

The industrial instances were provided by the Brazilian company whose production scheduling problem motivated this work. The parameters of each instance are in Table 10.

<table>
<thead>
<tr>
<th>Comb.</th>
<th>( L/L/J/J/T )</th>
<th>Comb.</th>
<th>( L/L/J/J/T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>5/9/33/11/1</td>
<td>B1</td>
<td>6/10/52/19/1/</td>
</tr>
<tr>
<td>A2</td>
<td>6/9/49/14/2</td>
<td>B2</td>
<td>6/10/56/19/2/</td>
</tr>
<tr>
<td>A3</td>
<td>6/9/58/15/3</td>
<td>B3</td>
<td>6/10/65/21/3/</td>
</tr>
</tbody>
</table>

Table 10 – Industrial instances

Notice that each combination represents now only one instance. The demands must be
satisfied at the end of each period. In the instances A, each period $T$ covers 7 days with 24 hours of available capacity per day. In the instances B, each period covers 10 days with 24 hours of available capacity per day. Table 11 shows the setup values.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line setup time (hours)</td>
<td>$\sigma_{ijl} = \begin{cases} 0.5 &amp; i \neq j \ 0.0 &amp; i = j \end{cases}$</td>
</tr>
<tr>
<td>Line setup cost ($$/\text{u})</td>
<td>$s_{ijl} = \begin{cases} 3000 &amp; i \neq j \ 0.0 &amp; i = j \end{cases}$</td>
</tr>
<tr>
<td>Tank setup time (hours)</td>
<td>$\sigma_{ijl} = \begin{cases} 1 &amp; i \neq j \ 0.5 &amp; i = j \end{cases}$</td>
</tr>
<tr>
<td>Tank setup cost ($$/\text{u})</td>
<td>$s_{ijl} = \begin{cases} 12000 &amp; i \neq j \ 6000.0 &amp; i = j \end{cases}$</td>
</tr>
</tbody>
</table>

Table 10 – Setup values

The inventory cost of product ($h_j$), the production costs for products ($v_{jl}$) and raw materials ($v_{jk}$) keep the values used in moderate-to-large sized instances. There is no value for the inventory cost of raw material ($\overline{h}_j$=0). All these cost values were used to evaluate the production plan really executed by the company for each instance. The values obtained determined the cost of the company schedules ($Z^I$) in Table 11. The GA ran 3 times over each instance with 3600 seconds assigned to each run. The GA parameters are the same used to solve moderate-to-large sized instances. Table 11 compares the cost of the company schedule ($Z^I$) with the better GA solution (GA-Min), the average GA solution (GA-Avg) and the worse GA solution (GA-Max).

<table>
<thead>
<tr>
<th>Comb.</th>
<th>$Z^I$</th>
<th>GA - Min</th>
<th>GA-Avg</th>
<th>GA-Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1692098.4</td>
<td>1662098.4</td>
<td>1667098.4</td>
<td>1671098.4</td>
</tr>
<tr>
<td>A2</td>
<td>3511909.7</td>
<td>3381356.8</td>
<td>3388058.8</td>
<td>3398509.8</td>
</tr>
<tr>
<td>A3</td>
<td>5002677.3</td>
<td>4837433.0</td>
<td>4847207.3</td>
<td>4859924.0</td>
</tr>
<tr>
<td>B1</td>
<td>3378204.8</td>
<td>3303499.5</td>
<td>3317499.5</td>
<td>3345499.5</td>
</tr>
<tr>
<td>B2</td>
<td>4278520.7</td>
<td>4174521.8</td>
<td>4199462.6</td>
<td>4222860.0</td>
</tr>
<tr>
<td>B3</td>
<td>7943402.2</td>
<td>7735818.0</td>
<td>7796636.3</td>
<td>7839039.0</td>
</tr>
</tbody>
</table>

Table 11: The cost of the company schedules and GA solutions for industrial instances.

The GA solutions costs are always better then the cost of the schedule executed by the company. One aim of this work was to provide good solutions within time limit of 1 hour. Of course, the GA might be find better results spending more than 1 hour of execution time. The schedule provided by the company is prepared using the experience of the decision maker. However, the GA was able to beat the industrial plan cost, at least in these industrial instances.

4. Conclusion

This paper presents an evolutionary approach to solve the Synchronized and Integrated Two-Level Lot Sizing and Scheduling Problem (SITLSP). The mathematical model proposed in Kimms et al. (2005) and Toledo (2005) is large and complex. This became hard for an exact method to find optimal solutions even for small-to-moderate size and moderate-to-large size instances. This paper presented the Synchronized and Integrated Two-Level Lot Sizing and Scheduling Problems (SITLSP). The software GAMS/CPLEX was used to obtain optimal solutions or best feasible solution at least. This approach was able to return optimal solution only for the small size instances with macro-periods $T=1$ and 2. Only best feasible solutions were returned for small-to-moderate size instances with macro-periods $T=3$ and 4. The exact approach was not able to return at least a feasible solution for large size instances. A Multi-Population Genetic Algorithm was proposed. This approach has a new representation of solutions and deals with a hierarchically structured multi-population. A tailor-made decoding procedure is used to evaluate the solution encoded in the gene of each individual. Moreover, a recombination over population clusters takes place and migrations among different populations are allowed. In the small-to-moderate size instances, the GA found many optimal solutions or it had a small average
deviation from the best feasible solutions returned by GAMS/CPLEX. In the moderate-to-large size instances, the GA found and improved solutions in a reasonable execution time while the exact approach didn't even find a feasible solution. In the industry instances, the GA was able to return better solution costs than the production plan executed by the industry. The results for large and industrial instances suggest that this evolutionary approach shows up as an alternative to solve real-world instances.

Referências


Kimms, A., Toledo, C.F.M., França, P.M. (2005), Modelo Conjunto de Programação da Produção e Dimensionamento de Lotes Aplicado a uma Indústria de Bebidas, Anais do XXVII SBPO.


