A Parallel Computing Approach to Solve the Two-Level Synchronized and Integrated Lot sizing and Scheduling Problem

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

Lucas de Oliveira
Departamento de Ciência da Computação – Universidade Federal de Lavras
Campus Universitário, C.P. 3037, CEP 37200-000, Lavras, MG
oliveiralukas@yahoo.com.br

Marluce Rodrigues Pereira
Departamento de Ciência da Computação – Universidade Federal de Lavras
Campus Universitário, C.P. 3037, CEP 37200-000, Lavras, MG
marluce@dcc.ufla.br

Paulo Morelato França
Departamento de Matemática, Estatística e Computação – Universidade Estadual Paulista
R. Roberto Simonsen, 305, CEP 19060-900, P. Prudente, SP
paulo.morelato@fct.unesp.br

ABSTRACT

This paper describes the use of parallel computing as an approach to solve the Synchronized and Integrated Lot sizing and Scheduling Problem (SITLSP). The multi-population genetic algorithm previously proposed to solve the SITLSP is adapted to be executed in parallel using a workstation with four processors. Three processors are responsible to evolve separately each population and a fourth processor manages the migration and restart operations among these populations. Thus, the parallel approach allows evolving simultaneously the populations of the genetic algorithm. Computational results are reported using industrial instances already available for the SITLSP and other complex instances proposed here. The parallel approach results are compared with those returned by the sequential multi-population genetic algorithm.


Main area Metaheurística
1. Introduction

Nowadays, computers with more than one processor have becoming ordinary among computer users. Also, multi-core platforms have becoming cheaper and more companies are able to acquire this kind of technology. In this way, solution methods for complex industrial problems can be tailor-made to take advantage from this new scenario using parallel computing.

Parallel computing aims to achieve high performance handling the available machines in a company and exploiting efficiently the parallelism from the algorithms. This kind of computing has been largely employed in the design of airfoils; internal combustion engines; high-speed circuits; sequencing of the human genome; simulation of earthquakes; data mining and analysis for optimizing business and marketing decisions; security and intrusion detection in computer systems, etc. (Grama et al., 2003, and Hughes and Hugher, 2004).

This paper presents an approach using parallel computing to solve the Synchronized and Integrated Lot sizing and Scheduling Problem (SITLSP). The SITLSP is motivated by an industrial problem usually found in soft drink companies. In these companies, there is a tank level with decisions about the amount and the timing that raw materials (soft drink flavor) have to be stored. There is also a production line level with decision about the amount and the timing which final products (soft drinks) must be processed by lines (Figure 1).

A synchronization problem occurs because the production in lines can not happen without the corresponding raw materials scheduled in tanks. A sequence-dependent setup time is necessary to clean and fill up a tank and no raw material can be pumped from this tank to a production line. If a line has two different products switched, there is also a sequence-dependent setup time. Each line produces a set of products and a same product can be processed in more than one line.

The SITLSP has two Capacitated Lot Sizing and Scheduling Problems (CLSP) as sub-problems (one in the tank level and another in the line level) which must be synchronized. The CLSP by itself is proven to be a NP-hard optimization problem by Bitran and Yanasse (1982). Discussion and reviews on the CLSP and other lot sizing and scheduling problems can be found in Karimi et al. (2003). A mathematical model for the SITLSP was first proposed by Toledo et al (2007), where the branch & cut algorithm faced problems to find optimal solutions even for small-to-moderate size instances.

A multi-population genetic algorithm with hierarchical tree structures has been proposed to deal with the SITLSP (Toledo et al. 2008). Genetic Algorithms (GAs) are evolutionary computational methods that simulate biological processes (Holland, 1975). GAs have been used to solve real-world problems which include lot sizing and scheduling problems (Jans and Degraeve, 2007).

concurrently dealing with different objectives. An overview in parallel computing and evolutionary methods is presented by Alba and Tomassini (2002).

This paper presents results using parallel computing to solve SITLSP. The next section describes the sequential GA tailor-made to solve the SITLSP and section 3 presents the parallel version of this algorithm. The computational results are reported on section 4 and the conclusions are shown on section 5.

2. Sequential Genetic Algorithm

This section summarizes the basic issues of the genetic algorithm (GA) previously proposed by Toledo et al. (2008) to solve the SITLSP (Figure 2).

```
Method Multi-Population GA
begin
repeat
  initializePopulations();
  repeat
    for i = 1 to numberOfPopulations do
      crossoverPop(i);
      mutatePop(i);
      structurePop(i);
    until convergence;
    migration();
  until time limit;
end
```

Figure 2 - Multi-population Genetic algorithm.

In this multi-population pseudo code, the genetic operators (crossover and mutation) are executed for each population. The populations converge when new individuals are not inserted in any population. At this point, a migration and initialization procedures take place. The next initialization keeps only the best individual and the individuals migrated. A problem solution (individual) is represented in this GA as a two-dimensional matrix with $T$ (number of periods) rows and $N$ columns (Figure 3).

![Figure 3 - Representation of solution.](image-url)
Each entry \((t,n), t \in T \text{ and } n \in N\), has the following data encoded in it:

- \(P_{t,n}\): product in position \(n\) to be produced in period \(t\).
- \(D_{t,n}\): lot size of product \(P_{t,n}\).
- \(SL_{t,n}\): sequence of lines where \(D_{t,n}\) can be produced.
  \(SL_{t,n} = (\alpha_1, \ldots, \alpha_k)\).
- \(STK_{t,n}\): sequence of tanks where the raw material of \(D_{t,n}\) can be stored.
  \(STK_{t,n} = (\beta_1, \ldots, \beta_k)\).

The demand \(d_{i,t}\) of product \(i\) in period \(t\) is split in several lots \((D_{t,n})\) and randomly distributed among the entries in periods \(t_1, t_2, t_3\). The sequences \(SL_{t,n}\) and \(STK_{t,n}\) are randomly generated with length \(k\). The sequence \(SL_{t,n} = (\alpha_1, \ldots, \alpha_k)\) with \(\alpha_i \in \{1, \ldots, L\}\), where \(\alpha\) is a possible line number and \(L\) is the number of lines. The sequence \(STK_{t,n} = (\beta_1, \ldots, \beta_k)\) with \(\beta_i \in \{1,2,\ldots,2L\}\), where \(\beta\) tells us where and how the raw material will be stored. The parameter \(\bar{L}\) is the number of tanks. The tank number \(j\) is obtained from \(\beta\) doing:

\[
j = \begin{cases} 
\beta_j, & 1 \leq \beta_j \leq \bar{L} \\
\beta_j - L, & \bar{L} < \beta_j \leq 2 \bar{L}
\end{cases}
\]  

(1)

If \(1 \leq \beta_j \leq \bar{L}\), the tank \(j = \beta\) 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 \(\bar{L} < \beta_j \leq 2 \bar{L}\), the tank \(j = \beta - \bar{L}\) will be immediately occupied. This forces the method to find solutions where the tank capacity is completely filled. These conditions have some exceptions. If tank \(j\), selected by one of the previous criteria, stores a raw material different from the raw material of the product \(P_{t,n}\), it will be necessarily occupied after the raw material previously stored has been used. The same will happen if \(j\) is completely full. On the other hand, if tank \(j\) is empty, it will be immediately occupied.

An example clarifies the solution representation. Suppose two products (P1 and P2) where each product has a demand of 100 units to be filled in period T1 and another 100 units to be filled in period T2. The products use different raw materials. Moreover, there are two lines and two tanks available to produce and store all these raw materials and products. Figure 4 gives two possible representations of solution.

![Solution 1](image1.png)

![Solution 2](image2.png)

Figure 4 - Examples of solutions representation.

The demands are distributed in their respective periods in solution 1, but the demand of P2 in T1 is split between two entries. In solution 2, part of the P1 demand in T2 is produced on T1. The sequence of lines \((SL_{t,n})\) and tanks \((STK_{t,n})\) can repeat values of \(\alpha_i \in \{1,2\}\) and \(\beta_i \in \{1,2,3,4\}\) for \(L=\bar{L}=2\) and \(k=4\) (length). A procedure discloses the information in each solution by decoding it into a useful problem solution (Figure 5).
Each pair \((\alpha_i, \beta_i)\) is selected from the sequence of values in \((SL_{t,n})\) and \((ST_{t,n})\) while \(D_{t,n}>0\). If \((\alpha_i, \beta_i)\) returns a line and a tank with available capacity, so \(D_{t,n}\) or a fraction of \(D_{t,n}\) is scheduled. If the capacity is not enough to produce \(D_{t,n}\) or it is enough to produce only a fraction of \(D_{t,n}\), the next pair \((\alpha_{i+1}, \beta_{i+1})\) is selected. These steps start with the first entry in the last period of the solution and they are repeated until reach the last entry in the first period. This backward process aims to postpone setups and processing time. However, there is no guarantee that all demands will be produced at the end.

The complete decoding of one solution returns a lot size and schedule for lines and tanks. This schedule is evaluated using the objective function of the problem. To sum up, the solution value is determined adding up all costs that happen in the final schedule: production, setup 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 value. The best solution is the one whose decoding process determines a final schedule with a minimum cost value.

The uniform crossover operator is applied once that it was the best crossover type for the SITLSP based on previous computational experiments (Toledo et al., 2008). For each entry in the same position, a random value \(\lambda \in [0,1]\) is generated. If \(\lambda < 0.5\), the new individual inherits the entry of one parent; otherwise, it inherits the entry of the other parent. It is not allowed excessive demand and a repair procedure fix demand deficits in some period.

The mutation operator executes three types of changes in the individual: swap the positions of two genes in the same period; swap the positions of two genes in different periods; and remove a gene inserting it in another position. The mutation type, genes and periods are all randomly selected. A mutation move can not violate demand satisfaction in some period.

A multi-population GA can allow a more effective exploration in the solution space because populations that evolve separately will present different characteristics according to the genetic drift idea (Winner, 1995). Also, the multi-population GA presents individuals hierarchically structured in binary trees. The computational results reported by Toledo et al. (2008) indicate a better performance for binary tree structures over ternary tree structures solving the SITLSP. It was adopted a binary structure with 5 clusters of 3 individuals each one arranged in three levels: 1 cluster in level 1, 2 clusters in level 2 and 4 clusters in level 3 (see Figure 6). It will be also considered in this paper 3 population with 15 individuals each one.
The crossover described previously is carried out over a cluster (selected at random) and always selects a supporter node (also randomly chosen) and its respective leader node (Figure 7). If the new individual (Child) is the better, it 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).

![Figure 7 - Keeping the hierarchical structure](image)

### 3. Parallel Genetic Algorithm

This section describes the parallel multi-population GA proposed for the SITLSP. The aim is to execute in parallel the independent parts of the sequential GA which have an execution bottleneck. The sequential GA described in the last section executes the genetic operators (recombine and mutation) and restructures population until the convergence takes place.

The populations converge when a fixed number of recombinates has been completed and no new individuals were inserted (Figure 2). At this point, a migration is executed, where a copy of each best individual is inserted into the next population (Figure 8). 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.

![Figure 8 - Migration process](image)

This paper proposes to execute in parallel these genetic operators for each population. It
is used a model related to the master-slave parallel algorithm model (To and Vohradsky, 2007). The master processing node is responsible to start the computation, coordinate the work distribution among the slave processing nodes, control the migration among these populations, update the best individual found and check the execution time.

The slave processing nodes are responsible to execute the genetic operators for its population and send to the master the best individual found when the population has converged. The migration is the step where the master and slaves nodes exchange information (Figure 9). This strategy is also similar to the master-slave message-passing paradigm used by Berger and Barkaoui (2004).

Figure 9 - The parallel processing

In order to explain better this strategy, consider the figure 9, where there are 4 processing nodes: 1 master and 3 slaves. There are 3 populations (one in each slave) and let’s suppose that the population 1 (slave 1) has converged. At this moment, a copy of its best individual is sent to the master processing node, where it will be compared with the best individual found so far. If the migrated individual is better, the best individual in the slave 1 is replaced. The migrated individual from population 1 is also stored by the master processing node until the population 2 (next population) has converged. Thus, it will be sent and inserted randomly into population 2.

At the same time, population 2 will send a copy of its best individual to the master processor which will compare it with the best individual and send it latter to population 3. In the same way, when population 3 has converged, it will receive the migrated individual from population 2. Thus, population 3 will send its best individual to the master processor which will also forward it to be inserted in population 1.

The master processing node is responsible to coordinate the executions and each population evolves in a different slave processing node. So, one population can converge faster than other sending many copies of the best individual found to the master processing node. In this case, the copies are compared with the one already stored in the master and only the best is kept to be inserted in the next population.

The master processing node also checks the execution time. When this time is exceeded, the algorithm execution is stopped. The architecture proposed in this paper presents parallelism at a population level which can be recognized as island or cellular model. However, the migration here will not take place after a fixed number of iterations as usually happen (To and Vohradsky, 2007). The migration will happen based on population convergence, where an
interface controlled by the master processor is responsible to manage this among populations.

4. Computational Results

The computational tests compare the parallel GA approach proposed by this paper with the sequential GA previously developed to solve the SITLSP. The methods are evaluated using industrial size instances. These instances were defined from real production schedules executed by a soft drink company. Each parameter combination in Table 1 defines one of these industrial instances. The parameters are the number of lines (L), tanks (Tk), products (Prod.), raw materials (RM) and periods (T). Each production period covers 7 days with 24 hours of production time for A instances and 10 days with 24 hour of production for B instances.

<table>
<thead>
<tr>
<th>Comb.</th>
<th>L</th>
<th>Tk</th>
<th>Prod</th>
<th>RM</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>6</td>
<td>9</td>
<td>49</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>A2</td>
<td>6</td>
<td>9</td>
<td>58</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>B1</td>
<td>6</td>
<td>10</td>
<td>56</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>B2</td>
<td>6</td>
<td>10</td>
<td>65</td>
<td>21</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1 - Industrial instances.

The setup time of products in lines are the same and takes 0.5 hours. The setup time for raw materials in tanks takes 1 hour to setup the same raw material and 2 hours to setup a different raw material. The setup cost is proportional to the setup time and it was adjusted as 1000*setup time. This value was chosen because it provides a suitable trade off between the different terms of the fitness function (Toledo, 2007). The costs related to the storage of raw material in tanks, the inventory cost of each product and the production process of each product and raw material were fixed in 1($/u), where u is the product or raw material unit. The tanks maximum and minimum capacities are limited to 1000l (liters) and 24000l, respectively. The demand range is 47-180000 units within the same period. The processing time range is 50-2000(u/hour) according to the product and line assignment.

The methods have been implemented in Java and they run in a workstation with: Windows Server 2003, Quad Core, Intel Xenon, 1.63 Gzh and 2Gb of RAM. The workstation did not execute other processes during the computational tests. The evolutionary approaches were set with 3 populations structured in binary trees of 15 individuals each one, with crossover rate of 1.5 and a mutation rate of 0.7. These parameter adjustments were based on previous computational results for the same problem (Toledo et al., 2008). The methods ran on during 1800 seconds per execution. A total of 10 executions took place over each instance. Table2 compare the average results found by the methods with the estimated cost (Z*) for the schedules executed in fact by the company. The average deviation from the estimated cost Dev(%) =100*(Z-Z*)/Z* is calculated, where Z is the average value of the final solution found in the 10 executions. The deviation values are also depicted.

<table>
<thead>
<tr>
<th>Executed by company</th>
<th>Sequential GA</th>
<th>Parallel GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comb.</td>
<td>Z'</td>
<td>Z</td>
</tr>
<tr>
<td>A1</td>
<td>3511.91</td>
<td>3365.73</td>
</tr>
<tr>
<td></td>
<td>-4.16</td>
<td>-3.86</td>
</tr>
<tr>
<td>A2</td>
<td>5002.68</td>
<td>4920.60</td>
</tr>
<tr>
<td></td>
<td>-1.64</td>
<td>-3.15</td>
</tr>
<tr>
<td>B1</td>
<td>4278.52</td>
<td>4160.84</td>
</tr>
<tr>
<td></td>
<td>-2.75</td>
<td>-2.85</td>
</tr>
<tr>
<td>B2</td>
<td>7943.40</td>
<td>7713.39</td>
</tr>
<tr>
<td></td>
<td>-2.90</td>
<td>-2.84</td>
</tr>
</tbody>
</table>

Table 2 - Average solutions found for A1, A2, B1 and B2.
Table 3 shows the average CPU time to reach the best solution found by the first time. It also shows the parallel GA average deviation from the sequential GA average time.

<table>
<thead>
<tr>
<th></th>
<th>Sequential GA</th>
<th>Parallel GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comb.</td>
<td>Time (sec.)</td>
<td>Time (sec.)</td>
</tr>
<tr>
<td>A1</td>
<td>1504.18</td>
<td>1492.23</td>
</tr>
<tr>
<td>A2</td>
<td>1172.83</td>
<td>1537.32</td>
</tr>
<tr>
<td></td>
<td>-0.79</td>
<td>31.08</td>
</tr>
<tr>
<td>B1</td>
<td>1785.37</td>
<td>1524.35</td>
</tr>
<tr>
<td></td>
<td>-14.62</td>
<td>-14.62</td>
</tr>
<tr>
<td>B2</td>
<td>1628.68</td>
<td>1399.61</td>
</tr>
<tr>
<td></td>
<td>-81.85</td>
<td>-81.85</td>
</tr>
</tbody>
</table>

Table 3 - Average time to find solutions for A1, A2, B1 and B2.

The parallel and sequential deviations are similar. It is not possible to take a conclusion about the parallel GA performance. These instances may not be complex enough to justify the use of parallel computing. Thus, another set of instances are proposed by this paper which will allow comparing the methods in a more complex situation.

The main idea is to create instances from A2 and B2 increasing the number of demand periods. These new instances will be called A3 and B3 with T=12, A4 and B4 with T=15, and A5 and B5 with T=18 periods. The demands in the first three periods of these new instances are identical to the demands available in the three periods of A2 and B2. However, the new periods T= 4—6, 7—9, 10—12, 13—15 and 16-18 will receive half of the demands in T=1—3. All the other instance parameters are kept.

The same idea was used by Kang et al. (1999) to evaluate their column generation methodology in a lot sizing and scheduling problem on parallel machines. The authors proposed new instances removing some products and coping demands to the new periods from the original demands in an industrial instance. Table 4-5 show the computational results found for these instances. It is also shown the average deviation from sequential GA solutions.

<table>
<thead>
<tr>
<th></th>
<th>Sequential GA</th>
<th>Parallel GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comb.</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>A3</td>
<td>9496.58</td>
<td>4238.35</td>
</tr>
<tr>
<td></td>
<td>-55.37</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>21992.49</td>
<td>15211.48</td>
</tr>
<tr>
<td></td>
<td>-30.83</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>44727.18</td>
<td>36093.27</td>
</tr>
<tr>
<td></td>
<td>-19.30</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>12607.43</td>
<td>3989.6</td>
</tr>
<tr>
<td></td>
<td>-68.35</td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>34399.12</td>
<td>12526.81</td>
</tr>
<tr>
<td></td>
<td>-63.58</td>
<td></td>
</tr>
<tr>
<td>B5</td>
<td>204235.09</td>
<td>103179.24</td>
</tr>
<tr>
<td></td>
<td>-49.48</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 - Average solutions found for A3, A4, A5, B3, B4 and B5.
<table>
<thead>
<tr>
<th>Comb.</th>
<th>Sequential GA Time (sec.)</th>
<th>Parallel GA Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3</td>
<td>1794.47</td>
<td>1629.79</td>
</tr>
<tr>
<td>A4</td>
<td>1615.06</td>
<td>1607.23</td>
</tr>
<tr>
<td>A5</td>
<td>1788.32</td>
<td>1790.50</td>
</tr>
<tr>
<td>B3</td>
<td>1796.79</td>
<td>1615.86</td>
</tr>
<tr>
<td>B4</td>
<td>1792.14</td>
<td>1610.22</td>
</tr>
<tr>
<td>B5</td>
<td>1618.61</td>
<td>1612.83</td>
</tr>
</tbody>
</table>

Table 5 - Average time to find solutions for A3, A4, A5, B3, B4 and B5.

The parallel GA outperforms the sequential GA in all these instances. The number of demand periods increased requiring more computational processing from the methods. In this context, the parallel approach had better average results than the sequential GA.

4. Conclusions

This paper presents a multi-population genetic algorithm with parallel computing as an approach to solve the Synchronized and Integrated Lot sizing and Scheduling Problem (SITLSP). This strategy is motivated by the search for good solutions and reduction of the long time spent to obtain them in large instances of this problem.

The parallel multi-population GA is designed to execute in simultaneously parts of the method which can represent an execution bottleneck. Therefore, it was implemented a master-slave model, where the master is responsible to coordinate the distribution of work between the slaves, update the best individual, migrate the best individual among populations and check the execution time of the method. The slaves execute the genetic operators over one population and send the best individual found to the master when the population has converged.

The results show that there is no significant difference between the parallel and sequential methods when applied to solve small instances. This happened with the available industrial instances A1, A2, B1 and B2. However, the parallel GA outperforms the sequential GA for the large instances proposed by this paper. These instances were obtained increasing only one of the problem parameters: the number of demand periods.

The computational results indicate that this strategy improved the method returning solutions with a better quality and faster than the sequential GA. The concurrent evolution of populations in different processors seems to improve the genetic algorithm performance.

The populations in the sequential GA converge when no new individual is inserted in any populations. Therefore, the method must wait this happen to execute migration and re-initialization of the populations. In this way, it seems that the crossover and mutation are executed very often in this approach, while the migration and re-initialization need to wait longer.

The populations in the parallel GA converge also when no new individual are inserted. However, if this happen in a population that evolves in one slave, the best individual is immediately sent to the master and the population is reinitialized. In this way, it seems that the crossover and mutation are executed less than in the sequential GA, while the migration and re-initialization occur very often. This can explain the better performance of the parallel GA in the large instances.

As future works, more computational tests will be executed aiming to evaluate better
the impact of migration and re-initialization in the parallel GA performance. It is also necessary to evaluate the parallel approach in instances with a larger number of products, raw materials, lines and tanks. At the moment, the parallel GA is being executed within a larger computational time to verify if more feasible solutions can be found.

Acknowledgment

This research was supported by Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), project CEX APQ-2271-5.01/07 and project EDT-91/07.

References


