A GENETIC ALGORITHM FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM (RCPSP)

UN ALGORITMO GENÉTICO PARA EL PROBLEMA DE PROGRAMACIÓN DE PROYECTOS CON RESTRICCIÓN DE RECURSOS (RCPPSP)

Edgar Gutiérrez Franco*, Fernando La Torre Zurita** y Gonzalo Mejía Delgadillo**

* School of Industrial Engineering, Universidad de La Sabana
** Department of Industrial Engineering, Universidad de los Andes, Bogotá - Colombia
ferltz@gmail.com
(Recibido el 25 de septiembre 2007, aceptado para publicación el 10 de noviembre 2007)

ABSTRACT
This paper proposes a Genetic Algorithm (GA) for the Resource Constrained Project Scheduling Problem (RCPSP). Resources are renewable and there is a unique way to perform the activities. This work employs Genetics Algorithms to schedule project activities to minimize makespan subject to precedence constraints and resources availability. A serial generation scheme is used to obtain the schedule. The algorithm was programmed using Object Oriented programming that allows generating individuals with their own attributes such as activity sequence and makespan. A Genetic Algorithm is proposed which uses a novel chromosome representation. The issues of the GA parameter tuning are also discussed in this paper. A computer tool that allows the user to define activities, precedence constraints and resource capacity was developed.

RESUMEN
En este artículo proponemos un Algoritmo Genético para el Problema de Programación de Proyectos con Restricción de Recursos (RCPSP por el acrónimo en inglés). Como los recursos son renovables, hay una manera única para llevar a cabo las actividades. En este trabajo se utilizan Algoritmos Genéticos para programar las actividades de un proyecto y minimizar su duración sujeta a restricciones de precedencia y disponibilidad de recursos. Se utiliza un esquema de generación en serie para generar la programación. Para la implementación del algoritmo se utilizó programación orientada a objetos, lo que permite generar instancias individuales con sus propios atributos e.g. secuencia de actividad y duración. Proponemos el uso de un Algoritmo Genético que utiliza una novedosa representación de cromosoma. Las cuestiones relativas al ajuste de parámetros del Algoritmo Genético también son discutidas en este artículo. Se desarrolló una herramienta informática que permite al usuario definir actividades, precedencia de restricciones y capacidad de recursos.

Keywords: RCPSP, Project Scheduling, Resource Constraints Projects, Genetics Algorithms.
Palabras Claves: RCPSP, Programación De Proyectos, Proyectos Con Restricción De Recursos, Algoritmos Genéticos.

1. INTRODUCTION
In project management, decisions depend directly on both quantity and quality of information and indirectly on the application method and technique used to schedule and control. So far, the time factor has played a huge role in scheduling and has always been related to its logic structure. Every project has its own precedence constraints, which means that each activity can be processed when all its predecessors are finished. In general, the purpose of Project Schedules is to minimize its completion time or makespan (C_max), subject to precedence constraints.

A more general version assumes that to develop one or more activities, resources such as tools, equipment, machines, or human resources, are needed. Each resource has a limited capacity. Consequently, at certain moments, one activity may not begin their processing due to resource constraints even if all their predecessors are finished. Problems of this type are well known as Resource Constrained Project Scheduling Problems (RCPSP). The RCPSP occurs frequently in high scale projects management such as software development, generation plants building, and military industry projects as design, development and building of nuclear submarines [1].

2. BIBLIOGRAPHICAL REVISION
This problem has been studied by many authors proposing different solution methodologies. Some authors are Bartschi [2] who applies a GA for the MRCPPSP solution, a multimodal scheduling problem. Colak et al. [3] solved the RCPSSP by using neuronal networks.
Mingozzi and Maniezzo [16] found lower bounds by means of a combination of techniques, integer programming and branch and bound. Hartmann [10] proposes a GA in which every gene of a chromosome is a dispatch rule, and he uses these genes as criteria to break ties inside the generation of sequences. Another important researcher is Hartmann [11], who models GA combining it with local search in case no improvements in solution are found, when passing from one generation to other. Kolisch [14] compares some priority rules with serial programming outlines and parallel programming and reports some experiences. Merckel et al. [15], solve this problem using ant colony technique (ACO), with good results. Ballestin [1] employs heuristic techniques and a GA variation denoted Hybrid Genetic Algorithm. Rangaswamy [18] solves the problem with Taboo Search (TS) and a two levels list. Jiang [13], propose the results of heuristics and metaheuristics techniques when solving RCPSP. Debels and Vanhoucke [4] created a GA called (BPGA), which generates two different kinds of populations using backward and forward sequence generation schemes.

3. RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM DESCRIPTION

Resource Constrained Project Scheduling Problem (RCPSP) consists on executing a group of activities subject to constraints. Precedence relationships force some activities to begin after the finalization of others. In addition, processing every activity requires a predefined amount of resources, which are available in limited quantities in every time unit. The RCPSP purpose is to find the starting times for the activities in a way that the difference between the finalization time of the latest activity and the start of the earliest sequenced activity is minimized. Blazewicz et al. (1983), referenced by Ballestin [1], proved that RCPSP, Job Shop classic generalization, belongs to the NP – hard type problems, which implies that heuristic algorithms are necessary to get good quality solutions for big instances.

RCPSP can be described as:

A project consists of a set of activities $V_i = (1, 2, ..., n)$. The activities 1 and n are dummy activities that represent the project start and end, respectively.

Activities are related by two types of constraints. First, precedence relations are “finish-start” (FS) type that is to say that j activity may just be started when i activity is concluded. For each activity $j$ there is a $Pred_j$ set of activities that must be finalized before $j$ can be started. The elements of $Pred_j$ are called $j$ immediate predecessors.

It is true, with no generality loss that:

$$Pred_j \neq \emptyset \forall j \neq 1 \land \forall j \neq n \exists i/j \in Pred_j$$

An activity $i$ is called a predecessor of $j$ if it exists,

$$i_0, i_1, ..., i_s, i_{s+1} \text{ with } i_0 = i, i_{s+1} = j,$$

$$i_b \in \text{Pred}_{i_{b+1}}, b = 0,1, ..., k$$

The $i$ activity is a $j$ (immediate) successor if, $j$ is an $i$ (immediate) predecessor. The $j$ immediate successors set are denoted with $Suc_j$. It is true that 1 (n) is predecessor (successor) of every activity. A second type of constraint is related to resources. Activities consume resources that are provided in limited quantities. A set of $K$ types of renewable resources is defined. A limited quantity of resource $k$ noted $R_k$ is available at every time unit. While being processed, an activity $j$ needs $r_{jk}$ units of $k \in K$ resource type during each period of duration $d_j$ (there is only one way of processing the activities). The activities cannot be interrupted once they are started.

The $d_j$, $r_{jk}$, $R_k$ parameters are supposed to be known, determined and integer. Also, $d_1 = d_n = 0$ and $r_s, k = r_{a_k} = 0$ for every $k \in K$.

The RCPSP problem instance is solved if the starting times (or ending times) for each activity satisfy the precedence and resources constraints, such that the duration of the project is minimized.

In Figure 1, an example of a project (denoted as PR1) is shown [1]. This project has $n = 7$ activities and $K = 1$ type of renewable resources with 2 units of capacity. The Figure has all the project data.
A GENETIC ALGORITHM FOR THE ...

In Figure 1, a possible sequence is represented with 8 units project duration. Figure 3 shows the optimal solution of the problem, with 5 units duration. The sequence is represented by a Gantt diagram. The horizontal axis represents time and the vertical axis represents utilization of a type of resource. If there were more than one resource, it is necessary to draw a diagram for every type of resource. Every rectangle represents an activity. The rectangle’s length represents activity duration, and the height shows number of units used by the activity.

**Figure 1 – PR1 project.**

**Definition**: $s_j(f_j)$ is noted as the start (end) of the $j$ activity in a sequence $S$ that denotes a vector of starting times $\{s_1, s_2, \ldots, s_n\}$ or, similarly a vector of ending times $\{f_1, f_2, \ldots, f_n\}$. $s_i=0$ and $s_n = \max\{f_i : i=1, \ldots, n-1\}$. $f_n$ is length or duration of project unless it is specified otherwise. If a sequence is feasible it will be noted as $T(s)$, and defined as a feasible solution, which meets all its precedence relationships and resource constraints.

An $S$ sequence given, $A(t) = \{ j \in V / s_j \leq t < f_j \} = \{ j \in V / f_j - d_j \leq t < f_j \}$ is an activity set being processed (active activities) at time $t$.

Conceptually, RCPSP can be modeled as follows:

![Figure 2](image1)

**Figure 2**

![Figure 3](image2)

**Figure 3**
Objective function (1) minimizes activity ending time that states projects end, therefore the project duration. Constraints (2) represent precedence relations, and constraints (3) limit resource utilization of every k resource at any moment t. Finally, constraints (4) define the decision variables as non negative integers. (1) – (4) is a conceptual model because A(t) set are functions of decision variables. Therefore, if the RCPSP is intended to be solved with integer mixed linear programming, another model must be used. RCPSP can be denoted as PS| prec | Cmax according to Pinedo’s notation.

RCPSP, however, becomes a solvable problem in a polynomial time if resource constraints are eliminated (previous conceptual model constraints (3.3)). The starting times for each activity are calculated as follows:

(i) \( s_j = 0 \);
(ii) \( s_j = \max \{ s_h + d_h / h \in \text{Pred}j \}; j = 2, ..., n \).

This method generates a sequence denominated ES (Earliest Start) sequence, where every activity is sequenced as early as possible, taking into account the precedence relationships. ESj is the starting time for j activity in ES sequence and EFj = ESj + dj the j activity latest ending time.

The opposite concept also exists. A sequence (possible with respect to the precedence relations) with the same project length as ES with each activity starting at the latest possible time. In this case the calculations or recursion are performed backwards, and the sequence is denominated LS (Latest Start):

(i) \( s_n = \text{ES sequence duration} \);
(ii) \( s_j = \min \{ s_j / j \in P_h \} - d_j, j = n-1, ..., 1 \).

\( s_j \) is obtained as LSj and LFj = LSj + dj the j activity latest ending time.

In Figures 4 and 5 the ES an LS sequences in correspondence with PR1 are represented, with 4 units duration.

The ES and LS sequences (or their equivalent, in forward and backward calculations) define a set of activities denoted as critical activities, whose \( ES_j = LS_j \) (in PR1 4, 5 and 6 activities). Every path in the graph whose length (measured by the sum of node durations that compose it) is the same as the ES (and LS) is defined as the critical path. The only critical path in PR1 is \{4, 5, 6\}. We will denote by CPM the common length of the critical paths. The CPM is a lower bound for the duration for the RCPSP. The deviation with respect to the CPM is used often for comparison among heuristics since, for large instances, the best solution is usually not known.
In the mathematical model proposed by Pinedo and Chao [17], \( x_{jt} \) is a variable with value 1 when \( j \) activity is completed in time \( t \) and 0 otherwise.

Then, \( i \) resource quantity needed by \( j \) activity during the \([t-1,t]\) interval is:

\[
R_{ij} = \sum_{a=t}^{t+P_{j}-1} x_{ja}
\]

If \( H \) is a makespan upper bound, a simple but not very tight value may be obtained using the following expression:

\[
H = \sum_{j=1}^{n} p_j
\]

Then, time to complete \( j \) activity is given by:

\[
\sum_{i=1}^{H} t \cdot x_{ij}
\]

and the makespan can be expressed as:

\[
\sum_{i=1}^{H} t \cdot x_{i+1,j}
\]

An Integer programming formulation can now be formulated as follows:

\[
\min \sum_{i=1}^{H} t \cdot x_{i+1,j}
\]

s.a.

\[
\sum_{i=1}^{H} t \cdot x_{ij} + p_j - \sum_{i=1}^{H} t \cdot x_{ij} \leq 0 \quad \forall j \rightarrow k \in A
\]

\[
\sum_{j=1}^{n} \left( R_{ij} \sum_{a=t}^{t+P_{j}-1} x_{ja} \right) \leq R_i \quad \forall i, t
\]

\[
\sum_{i=1}^{H} x_{ij} = 1
\]

Schedule objective is to minimize makespan. Set (5) of constraints assures that precedence restrictions are met, \( j \) activity is followed by \( k \) activity. Set (6) of constraints ensures that the total \( i \) resource demand at \( t \) time, cannot exceed resource availability. The third set of constraint ensures that every activity is processed.

When the number of activities is large and the planning horizon is long, the RCPSP is usually solved using heuristics and metaheuristics, which proved already to provide effective solutions.

4. OBJECT ORIENTED MODEL

The RCPSP problem was modeled by a class called “individual”, which is represented in Figure 5. This class contains four attributes:

- Activities Sequence: Array that contains sequence of the activities.
- Starting Dates: Vector that contains starting dates of every activity respecting precedence restrictions.
- Ending Dates: Vector that contains the ending dates of every activity.
- \( C_{\text{max}} \): The value of the objective function. As shown in section 3, it is the largest completion time of all project activities.
In Figure 7 the individual’s structure that takes part in every generation at Genetic Algorithm is shown. It has been decided to use an objects vector to take advantage of Java tools as comparator interface for arrangement of every position and methods instead of the “sort” utility.

5. GENETIC ALGORITHM

GA, ‘Genetic Algorithms’ (Holland, 1975), is a search algorithm based on the natural selection mechanism and on genetics. It states an analogy between a set of problems to be solved and the set of individuals in a natural population. Solution information is codified in a string of numbers called chromosome. Potential solution chromosome is measured using an evaluation function (in which the problem objective function plays an important role), and it is called solution fitness. Pseudo algorithm GA is shown below:

Pseudo-Algorithm GA:

- Set t=0. Generate initial population $P_t$
- Evaluate the population $P_t$
- While stopping criteria is not satisfied
- [Select some elements from $P_t$ to copy in $P_{t+1}$]
- Perform crossover on selected elements of $P_t$ and move them to $P_{t+1}$
- Perform mutation on selected elements from $P_t$ and take them to $P_{t+1}$
- Evaluate the new population $P_{t+1}$
- $P_t = P_{t+1}$

Main elements that characterize a GA application are solution codification, crossover operators that are used and evaluation function. Other important decisions that may be taken into account to construct a GA are the initial population, mutation operators, parents’ selection rules and elimination of one or more individuals and the algorithm’s termination criteria

5.1 Schedule Generation Schemes

Schedule Generation Schemes (SGS) are a very important part of many procedures in RCPSP. The SGS build a partial sequence, which initially sets the start 0 to the activity 1. A partial sequence is a sequence where only a subset of n activities has been scheduled. Two different SGS exist: Series and Parallel. Series uses the increase of number of
activities to make the steps, whereas Parallel uses the temporary increase. In this paper Serial SGS is the employed scheme, because it generates active programs, i.e. an operation cannot be moved backwards in time without moving one or more forward.

Serial SGS consists on g=1, ..., n-2 stages (n is the number of activities). In every stage an activity is selected and scheduled as early possible, respecting precedence relationship and resource constraints. There are two associated sets to every g stage. The Sec_g set has activities already scheduled and D_g is the set of selectable activities. F_g is a set of finalization times for all elements of set D_g. An activity is selectable if it has not been scheduled and all its predecessors have already been scheduled. Mathematically we can define D_g as follows:

\[ D_g = \{ j \in V \setminus \text{Sec}_g ; \text{Pred}_j \subseteq \text{Sec}_g \} \]

The union of these sets is not set V in general, since it can also have non programmed - non eligible activities.

**Series Pseudo – Algorithm:**

1. Beginning: \( s_i = f_i = 0 \), \( \text{Sec}_1 = \{1\} \).
2. From \( g = 2 \) until \( n-1 \), do:
   2.1 Calculate \( D_g, F_g, R_k(t) \) (\( k \in K \); \( t \in F_g \)).
   2.2 Select \( j \in D_g \).
   2.3 \( ES_j = \max_{k \in \text{PRED}_j} \{ f_h \} \).
   2.4 \( s_j = \min \{ t \mid t \geq ES_j \}; t \in F_g ; t_j \leq R_k(t) \forall k \in K, t \in [t,t+d_j] \cap F_g \} \).
   2.5 \( f_j = s_j + d_j \).
   2.6 \( \text{Sec}_g = \text{Sec}_{g-1} \cup \{ j \} \).
3. \( f_n = \max_{k \in \text{PRED}_n} \{ f_h \} \).

The first stage and the last stage are assigned as a dummy activities and both are included in a partial sequence. Beginning every stage \( g \) a set of selectable \( D_g \) ending times, set \( F_g \) and remaining availabilities \( R_k(t) \) are calculated. After that, an activity \( j \) is selected from selectable activities set, its starting time is calculated by determining first \( ES_j \), earliest starting time \( j \), according to precedence relations and its predecessor’s activities denoted by \( h \) in the partial sequence, and then, calculating the earliest starting time related \( ES_j \) resources. Ending time \( j \) is calculated by adding \( d_j \) to \( s_j \). Table 1 shows how the serial scheme generates the sequences showed in Figure 3.

**TABLE 1 - SERIAL GENERATION EXAMPLE**

<table>
<thead>
<tr>
<th>( g )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Sec}_g )</td>
<td>(1)</td>
<td>(1,4,5)</td>
<td>(1,4,5,6)</td>
<td>(1,2,4,5,6)</td>
<td></td>
</tr>
<tr>
<td>( F_g )</td>
<td>(0)</td>
<td>(0,1)</td>
<td>(0,1,2)</td>
<td>(0,1,2,4)</td>
<td>(0,1,2,4)</td>
</tr>
<tr>
<td>( D_g )</td>
<td>-2.4</td>
<td>-2.5</td>
<td>-2.6</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>( j )</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

A possible sequence \( S = (s_1, s_2, \ldots, s_n) \) is said to be left active or left justified if there are no activities that may be started before without retarding any other activity. Similarly, a possible sequence is said to be right active or right justified if there is no activity that can be started after without delaying any other activity or delaying the sequence duration. Kolisch [14] proved that Series SGS generates active sequences.

In PR1 the only optimal solution is Figure 4 sequence that is an active sequence with no delays. Series SGS is able to transform any list of activities in a possible sequence. To do that it is just necessary to change the 2.2 step “Select \( j \in D_g \)” “for \( j = j^*_g \)”, (Hartmann [9]). Series SGS transforms the activity list \( \lambda = (1 \ 4 \ 5 \ 2 \ 6 \ 3 \ 7) \) for PR1 in Figure 4 sequence. Parallel SGS may also apply, calculating a priority vector according activities list, by using \( p_i = \text{order} (i, \lambda) \) (order or position of \( i \) in \( \lambda \)), and selecting activity in \( D_g \) with the lowest priority level. This modification leads to selecting activities to schedule in an order not designated by the activities list. Therefore, Serial SGS is usually used as the
scheduling scheme when implementing this representation. With Serial SGS codification all active sequences can be obtained. In general, an active sequence admits more than one representation as activity list and redundancy is not controllable a priori.

5.2 Chromosome Representation

The chosen representation for the RCPSP is based on the concept of activities list that has shown better performances compared to other representations according to Hartmann [11]. In an activities list $λ=\{j_1,\ldots,j_J\}$ each non dummy activity $J$ appears exactly once. Only activities lists that meet precedence constraints are considered. An activities list is able to be created if the predecessor activities are executed always before any activity on the list. Formally it can be expressed as $P_j \subseteq \{0,j_1,\ldots,j_J\}$ for $i=1,\ldots,J$.

An individual is represented by:

$$I = (j'_1,\ldots,j'_J)$$

In the activities sequence, precedence constraints are supposed to be met, this can be expressed formally like this:

$$\{j'_1,\ldots,j'_i\} = \{1,\ldots,J\} \text{ and } P_{j'_i} \subseteq \{0,j'_1,\ldots,j'_{i-1}\} \text{ for } i=1,\ldots,J$$

Each chromosome is related to a program that is calculated by using serial SGS, the initial node that corresponds to a dummy activity, start in time 0. The following activities are scheduled in an order defined by list $(j'_1,\ldots,j'_J)$.

5.3 Crossover Operators

For activities list building two crossover operators of one and two points are considered.

5.3.1 One Point Crossover

Two individuals denoted as parents are selected 1 ($λ^1$), parent 2 ($λ^2$):

$$λ^1 = \{j^1_1,\ldots,j^1_J\} \text{ and } λ^2 = \{j^2_1,\ldots,j^2_J\}$$

Then, a q number is selected randomly between 1 and J, to obtain two new individuals son 1 ($h^1$) and son 2 ($h^2$). Position $i=1,\ldots,q$, in $h^1$ is taken from $λ^1$ as follows:

$$h^1(j_i) := λ^1(j_i)$$

Activities from $i=q+1,\ldots,J$ positions in $h^1$ are taken from $λ^2$, however works already taken from $λ^1$ can not be considered again. Then it is obtained:

$$h^1(j_{i-k}) := λ^2(j_k)$$

where k is the smallest index $λ^2(j_k) \notin \{h^1(j_1,\ldots,j_{i-k})\}$.

As a result, relative positions on parent’s list are preserved. In this way parents helps their offspring objective function.

To illustrate this in a better form see next example: $λ^1 = \{1,3,2,5,4,6\}$, $λ^2 = \{2,4,6,1,3,5\}$ with $q=3$, an individual son is obtained: $h^1 = (1,3,2,6,4,5)$.

5.3.2 Two Point Crossover

It is an extension of one point crossover where two integer numbers $q_1$ and $q_2$ are chosen randomly with $1 \leq q_1 < q_2 \leq J$. Now $h^1$ son is determined by activities list on the $i=1,\ldots,q_1$ of $λ^1$ positions as follows:

$$h^1(j_i) := λ^1(j_i)$$
Activities on positions $i=q_1+1$, ..., $q_2$ in $h^1$ are taken from $\lambda^2$, by:

$$h^1(j_i) := \lambda^2(j_k)$$

where $k$ is the smallest index from: $\lambda^2(j_k) \notin \{h^1(j_1, ..., j_{i-1})\}$

Positions $i=q_2+1$, ..., $J$, are taken again from $\lambda^1$ : $h^1(j_i) := \lambda^1(j_k)$ where $k$ is the smallest index from: $\lambda^1(j_k) \notin \{h^1(j_1, ..., j_{i-1})\}$.

For this example, the new individual would be: $h^1=(1, 2, 4, 3, 5, 6)$

Son chromosome is formed by first and third part from parent 1 and second part from parent 2.

### 5.4 Mutation

Given the chromosome based on $\lambda$ activities list, the mutation operator modifies structure of the chromosome in the following way: for every position $i=1, ..., j-1$, activities $j_i$ and $j_{i+1}$ are exchanged with a defined probability, if the result is an activity list that meet the precedence restrictions.

Operators of mutation reported in technical literature were proved to create activities lists. However, it must be pointed out that mutation does not necessarily imply a change in schedule related to the chromosome, because redundancy may exist in schedule representation.

### 5.5 Evolution Strategy

Employed strategy is named “elitist” and it consists of selecting the best individual and generating an important number of individual of next generation from them, while other portion is generated by all individuals.

### 5.6 Fitness Function

The objective function to minimize is the ending time of all activities which is makespan ($C_{\text{max}}$). The fitness function is calculated as $1/C_{\text{max}}$.

### 6. RESULTS

The RCPSP instances were solved by using a program built in Java language. The program was run in a Pentium IV 3.0 MHz speed processor. Two instances were run from PSLIB library, from http://129.187.106.231/psplib/ URL. The sizes of the instances were 60 and 30 activities with 4 limited resources, and each activity with a maximum of 3 successors. The following tables show the parameters for the algorithm.

<table>
<thead>
<tr>
<th>TABLE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Population size</td>
</tr>
<tr>
<td>Parents percent</td>
</tr>
<tr>
<td>Sons percent</td>
</tr>
<tr>
<td>Cmax average</td>
</tr>
<tr>
<td>Run average time</td>
</tr>
<tr>
<td>Optimal</td>
</tr>
<tr>
<td>Standard deviation</td>
</tr>
</tbody>
</table>
In Table 2 and 3 results for 30 activities project are shown. In this project method varies in the crossover method by two and one point respectively, showing best results with two-point crossover method, even if its execution average time was higher due to larger number of calculations.

**TABLE 3**

<table>
<thead>
<tr>
<th></th>
<th>30 ACTIVITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Parents percent</td>
<td>0.3</td>
</tr>
<tr>
<td>Sons percent</td>
<td>0.7</td>
</tr>
<tr>
<td>Crossover</td>
<td>1 point</td>
</tr>
<tr>
<td>$C_{\text{max}}$ average</td>
<td>46.4</td>
</tr>
<tr>
<td>Run average time</td>
<td>3.2885</td>
</tr>
<tr>
<td>Optimal</td>
<td>43</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.86036613</td>
</tr>
</tbody>
</table>

In instances shown in Tables 4 and 5, a more complex project instance was tested. The same value of $C_{\text{max}}$ that is referenced in PSLIB was found here (The optimal value is not specified).

**TABLE 4**

<table>
<thead>
<tr>
<th></th>
<th>60 ACTIVITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Parents percent</td>
<td>0.3</td>
</tr>
<tr>
<td>Sons percent</td>
<td>0.7</td>
</tr>
<tr>
<td>crossover</td>
<td>2 points</td>
</tr>
<tr>
<td>Run average time</td>
<td>34.9017</td>
</tr>
<tr>
<td>$C_{\text{max}}$ average</td>
<td>85</td>
</tr>
</tbody>
</table>

**TABLE 5**

<table>
<thead>
<tr>
<th></th>
<th>60 ACTIVITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Parents percent</td>
<td>0.3</td>
</tr>
<tr>
<td>Sons percent</td>
<td>0.7</td>
</tr>
<tr>
<td>crossover</td>
<td>1 point</td>
</tr>
<tr>
<td>Run average time</td>
<td>29.3294333</td>
</tr>
<tr>
<td>$C_{\text{max}}$ average</td>
<td>85</td>
</tr>
</tbody>
</table>

It is easy to see that the objective function experiments a significant improvement in the first iterations and then it stays on a local optimum for a large number of iterations. This behavior is normal in Genetic Algorithms working with a high diversification level, generating at first a large search space and finding good solutions at the end. The search is intensified by the use of elitism to generate little improvements in final generations.
7. CONCLUSIONS

Tests were made with different mutation probability values and even though there were some variations in the schedule, no significant changes were produced in the objective function. This was mainly because redundancy exists, which means that two different schedules with different sequences may have the same $C_{max}$ value. Use of object oriented programming made easier to manipulate cromosomes and generations in the Genetic Algorithm. The two-point crossover provided better results even at the expense of longer computational times. In the instance J60 that has a greater number of activities, the resulting $C_{max}$ was the same for different generations. This is due to the many precedence and resource capacity restrictions such an instance has. Thus it is possible to see the lack of freedom that the genetic operators have in such a restricted problem as RCPSP.

8. FUTURE RESEARCH

This work presents an object oriented model to generate a project scheduling and a computational tool to support resources decision. We will use this technique to solve more general problems as multimode project scheduling problems. We want also to try more instances to calibrate parameters as cross operator, population size and generations number.

9. REFERENCES


