Managing Project Activities System using Genetic Algorithm

MILENA KAROVA, GERGANA TODOROVA, IVAYLO PENEV, MARIANA TODOROVA
Department of Computer Science and Engineering
Technical University of Varna
Studentska Str. 1, 9010 Varna
BULGARIA
milena.karova@gmail.com

Abstract: - A Genetic Algorithms technique is used to optimize project schedule in the present paper. The project schedule is created in Microsoft Project. The developed model is called OPTPROJECT. It is simple and at the same time general enough for optimization of projects, where the high cost activities have to be performed at the end of the project. The proposed application can be used to manage both small and large projects.

Key-Words: - Genetic Algorithms, Microsoft Project, Optimization Problem, Software Project Management

1 Introduction

A project is a series of tasks that will culminate in creation or completion of some new initiative, product or activity by a specific goal. Usually the projects are connected with an implementation of a strategic goal of the organization. Therefore the project management is of essential importance for each company. It is particularly important at the beginning of a project when the managers must answer the questions what, when, by whom and for how much money needs to be done.

Optimize a project schedule is a NP-hard problem, meaning that there are no known algorithms for finding optimal solutions in polynomial time. During the recent years this problem is discussed in a set of articles [1, 2, 4]. Many strategies can be used to solve the problem, including heuristic methods and genetic algorithms. Evolutionary algorithms are often used to conquer NP-hard problems. The studies [4, 10] show that the genetic algorithms are more suitable for schedule optimization than differential evolution. They are better on both speed and quality of the optimization.

Genetic algorithm (GA) [9, 11, 12, 13, 14] is travelling in a search space using more individuals so that they are less likely to get stuck in a local extreme like the other methods. They are also easy to implement. However, for some problems, choosing and implementation of encoding and fitness function can be difficult. The disadvantage of genetic algorithms is in the computational time. Genetic algorithms can be slower than other methods. But since the computation can be terminated at any time, the longer run is acceptable, especially with faster computers.

The problem solved in this paper is the application of a genetic algorithm for optimization of a project schedule created in Microsoft Project. MS Project is widely used in industry to set up a project schedule but its optimization is done manually assigning a priority number to a task. The developed application OPTPROJECT provides an interface that allows the user to read the XML file created with MS Project, management of GA, obtaining the optimal solution for every performance of GA, output the value of the best – the fitness function and the optimal policy for the implementation of the project activities. The application is implemented using programming language C# and development environment Visual Studio. Net 2008. The UML class diagram is given on Fig. 10.

OPTPROJECT can be used to manage both small and large projects of each area of activity, for which the high costs are necessary to carry out at the end of the project.

2 Definition of Project Management Problem

If a list of tasks with their duration, relations and constraints, and also a list of available resources and budget for the project are given, the objective of project scheduling is to find such plans that satisfy task durations and relations, resource constraints and budget limits. This problem is known as the Resource Constrained Project Scheduling (RCPS). In RCPS tasks can use multiple resources, and resources can have a capacity larger than one.
The usual optimization problem is to minimize the project duration or the project cost.

The case when the optimization goal is to schedule tasks within given deadlines, so that the cost for the project should be minimal will be considered, i.e. the tasks with highest costs need to be scheduled as late as possible. It is necessary to find such a set of schedules, so that the contractor has to pay for the resources as close as possible to the deadline of the whole project.

A set of constraints has to be met for the solution to be valid:
- The project has to be finished by the deadline.
- It is not important that the tasks have to be finished by their deadline; it is however important to meet the project deadline.
- The resources must not be overbooked.
- Task relations have to be preserved.
- Costs for material resources appear at the beginning of the task and costs for work resources at the end of the task.
- The dates in the projects are not important; it is the duration of the task.
- Costs for the project have to be paid during the project, but the contractor gets paid at the end of the project.
- Working resources can be grouped by their skills and every group can have its own price for a time unit, however the price for a resource within a group is the same.

The project schedule is often created in MS Project by a Gantt chart, which shows when each activity is performed. MS Project supports a variety of import and export formats. XML is a self-defining, adaptive language that is used to define and interpret data between different applications, particularly in Web documents. The simple and consistent nature of XML makes it very useful for exchanging data between many types of applications.

The main goals to achieve the improvement in scheduling with MS Project are evaluate export from MS Project and find a good optimization algorithm for the given problem.

3 Model Implementation

The Resource Constrained Project Scheduling (RCPS) has been proven NP-hard. Genetic algorithms showed good results in RCPS, therefore genetic optimization algorithm is offered to solve this problem.

Genetic algorithm [5, 6, 7, 8] provides stochastic search over the parameter space, guided by fitness evaluation towards specific goal. Although relatively slow, GA can handle complex optimization problems, especially when the goal is to find near-best extreme in multi-modal function domain. Genetic algorithm features also global search from many parameter space points and capability to escape from local extreme.

The structure of the realized genetic algorithm is shown on Fig. 1.

4 Chromosome Encoding

A genotype I is represented by a task list. To create a genotype it starts with a source task, which actually will not be part of the schedule. Each subsequent task is selected randomly among all tasks whose predecessors have already been selected in the sequence of tasks thus maintaining the link between tasks. The selected tasks are marked to not be reused.

Each gene in chromosome carries information about:
- UID – unique identifier;
- s - payments on the start of a task;
- e - payments for working resources in a task;
- d – duration of a task;
- Name – name of a task;
- Predecessors – a predecessor UID list;
- Milestone – indication of whether the activity is a task or an intermediate stage (milestone).

All variables, except the Name and Milestone, are integers. Name is a string, and the Milestone is a Boolean variable.
Let us consider a project [4] represented by the graph (Fig.2) and the Gantt chart (Fig.3.), which shows when each activity is performed. The project contains two smaller sub-projects. The first sub-project contains tasks B, C, D, E, F and the other one contains tasks G, H, I, J, K, L, M.

The chromosome chart of Project 1 is shown on Fig.4.

5 Initialization

Initial population contains a number of task lists created the way described above. The task list controls itself at the end of the creation process, so all the tasks in the population meets the constraints. Population size $N$ is set by the user in the range from 20 to 100.

6 Fitness function

The main goal of the optimization is to schedule “heavy” tasks later. The fitness function is the sum $S$ of $f_t$ in the task sequence.

$$f_t = s_t * p_t + e_t * d_t * p_t, \quad (1)$$

where $p$ is the position in the task list.

$$S = \sum_{i=1}^{T} f_i \quad (2)$$

The fitness function $S$ is calculated for each chromosome by using (2). For example to compute $S$ of the task sequence $I_1$ (G H L I M J K) in Fig.5, first $f$ has to be determined for each task:

$$S = \sum_{i=1}^{7} (s_i * p_i + e_i * d_i * p_i) =$$

$$= (3.1 + 16.4.1) + (1.2 + 12.6.2) +$$

$$+ (4.3 + 24.4.3) + (0.4 + 2.1.4) +$$

$$+ (1.5 + 8.1.5) + (0.6 + 0.0.6) +$$

$$+ (4.7 + 24.6.7) = 1602 \quad (3)$$

Another individual $I_2$ is the task sequence G L H M I J K (Fig.5) where $S$ is:
\[ S = \sum_{i=1}^{7} (s_i * p_i + e_i * d_i * p_i) = \\
= (3.1 + 16.4.1) + (4.2 + 24.4.2) + \\
+ (1.3 + 12.6.3) + (1.4 + 8.1.4) + \\
+ (0.5 + 2.1.5) + (0.6 + 0.6.6) + \\
+ (4.7 + 24.6.7) = 1568 \]  

(4)

Hence \( I_1 \) is a better solution than \( I_2 \).

The objective is to find a chromosome-feasible solution that maximizes the fitness function \( S \) and satisfies the constraints specified above.

7 Genetic Operation

7.1 Crossover

The first procedure of the genetic algorithm is crossover. This procedure takes two individuals as input, parent 1 and parent 2, combines their genes and produces two new individuals called child 1 and child 2, i.e. the crossing probability is \( P_c = 1 \). A two-point crossover is applied for which \( q_1 \) and \( q_2 \) are random integers with \( 1 \leq q_1 < q_2 \leq J \), where \( J \) is a number of tasks in the task list.

The child 1’s task list is determined by taking the task list of the positions \( i = 1, \ldots, q_1 \) from the parent 1. The next tasks are determined by taking the task list of the positions \( i = q_2 + 1, \ldots, J \) from the parent 2. However, the activities already selected may not be considered again.

parent1:

\[ \begin{array}{c}
G \\
H \\
L \\
I \\
M \\
J \\
K \\
\end{array} \]

parent2:

\[ \begin{array}{c}
G \\
L \\
H \\
I \\
J \\
M \\
K \\
\end{array} \]

cchild1:

\[ \begin{array}{c}
G \\
H \\
L \\
I \\
J \\
M \\
K \\
\end{array} \]

cchild2:

\[ \begin{array}{c}
G \\
H \\
L \\
I \\
M \\
J \\
K \\
\end{array} \]

Fig. 6. Example of a two-point crossover (\( q_1=3; \ q_2=6 \))

7.2 Mutation

Mutation is applied to every new child, produced by the crossover operator. It modifies the genes of the genotype with probability \( P_m \). The user may enter the value of the probability from 0 to 0.99. The mutation strategy is: it is gone through the individual’s task list from left to right and activities \( j_i \) and \( j_{i+1} \) are swapped with probability \( P_m \), where \( i \) runs from \( l \) to \( J-l \) (Fig. 7, Fig. 8). The swap is executed only if the resulting task is precedence feasible. That is \( j_i \) and \( j_{i+1} \) are not swapped if \( j_i \) is a predecessor of \( j_{i+1} \).

child1 before mutation:

\[ \begin{array}{c}
G \\
H \\
L \\
I \\
J \\
M \\
K \\
\end{array} \]

child1 after mutation

\[ \begin{array}{c}
G \\
H \\
I \\
L \\
J \\
M \\
K \\
\end{array} \]

Fig. 7. Example of mutation of a child 1

child2 before mutation:

\[ \begin{array}{c}
G \\
L \\
H \\
I \\
M \\
J \\
K \\
\end{array} \]

child2 after mutation

\[ \begin{array}{c}
G \\
H \\
L \\
I \\
M \\
J \\
K \\
\end{array} \]

Fig. 8. Example of mutation of a child 2

7.3 Parent selection

The best individuals from both old and newly generated task lists are selected as input for next generation. The ranking method is used, i.e. the best \( N \) individuals are chosen from a population of \( 2*N \) as parents for next cycle (Fig. 9).

parent1:

\[ \begin{array}{c}
G \\
H \\
L \\
I \\
M \\
J \\
K \\
\end{array} \]

\( S = 1700 \)

\[ \begin{array}{c}
G \\
H \\
L \\
I \\
M \\
J \\
K \\
\end{array} \]

\( S = 1602 \)

\[ \begin{array}{c}
G \\
H \\
L \\
M \\
I \\
J \\
K \\
\end{array} \]

\( S = 1595 \)

Fig. 9. Example of selection

Genetic algorithm evolution stops if reaches the specified maximum number of generations. The program provides the user possibility to set the maximum number of generations in the range from 10 to 100.

8 Description of the developed application OPTPROJECT

OPTPROJECT provides an interface that allows the user to obtain the optimal policy for the implementation of the project activities.

The input data for OPTPROJECT are: population size, number of generations, probability of mutation, a list with knowledge about the project (the project activities, the links between them and the constraints) as .XML file exported from MS Project.
The output data from the application are: the best chromosome got by executing the algorithm and presented in tabular form; the maximum value of the fitness function and the change of the fitness function based on the number of generations, the probability of mutation and the population size.

The following classes are created in the developed application OPTPROJECT:

8.1 Class TaskGene
This class represents the gene for the task and contains the member variables: UID, s, e, d, Name, Predecessors and Milestone. It has three constructors: a default constructor, explicit constructor and copy constructor.

Default constructor sets UID, s, e and d to initial value zero; Name - empty string; Predecessors - value null; Milestone - value false.

The explicit constructor receives as parameters the data for the task and saves them in the relevant member variables. The copy constructor receives as a parameter the source gene and copies its member variables in its own member variables.

8.2 Chromosome class
This class contains a member variable Genes, which is the class List <TaskGene>. The class has a default constructor that initializes Genes as an empty list and a copy constructor that receives as a parameter the source chromosome and copies its member variable Genes in its own member variable.

Chromosome class has two member functions: FitnessFunc and SwapGenes.

The FitnessFunc member function calculates the fitness function and returns the result as type decimal.

The SwapGenes member function receives as parameters the indices of genes i and j, and swaps locations of the genes.

8.3 Class Chromosome Comparer
This class inherits the interface IComparer <Chromosome>. It also contains the function IComparer <Chromosome>. Comparer, which receives as parameters x and y chromosomes and compares their target functions. The function returns -1, if the result of the fitness function x is greater than the target function of y. It returns 1 result in the opposite case, and 0 if the fitness functions are equal.

8.4 Class FitnessForm
This class inherits the class Form. It is used to plot the graph of the relationship between the fitness function and the number of generations.

8.5 Class MainForm
This class also inherits the class Form. It contains the form with the user interface.

Fig. 10 shows a UML class diagram.

Measures are taken in program OPTPROJECT to ensure the correctness of the entered data. Only valid numbers can be entered in the text fields for the population size and the probability of mutation. The number of generations is selected from the ComboBox, containing only valid values in the range from 10 to 100.

Fig. 10. UML class diagram

9 Test results
There are a variety of ways for evaluating and experimenting of the genetic algorithms. There is no single methodology for testing and evaluation. The most studies include subjective assessments based mainly on solved problems. Standard approach does not exist.

The present study keeps to the changing of the basic parameters of the genetic algorithm to solve the task of project management. The behavior of the algorithm is investigated and analyzed in case of change of the basic parameters: the number of generations, the number of chromosomes in the population and the probability of mutation Pm.

Two different tests are created. First the smaller schedule given on Fig. 2 is tested.

An analysis according to the basic parameters is made.

Some of the obtained results and the optimal order of execution of the tasks are given below.
optimal sequence of tasks for Project 1 (Fig. 11) and for Project 2 (Fig. 13) are presented.

Table 1. Experiments of OPTPROJECT changing the number of generations (Project 1)

<table>
<thead>
<tr>
<th>Number of generations</th>
<th>Best value of S</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2252</td>
</tr>
<tr>
<td>40</td>
<td>2353</td>
</tr>
<tr>
<td>60</td>
<td>2353</td>
</tr>
</tbody>
</table>

Table 2. Experiments of OPTPROJECT changing the chromosomes in the population (Project 1)

<table>
<thead>
<tr>
<th>Number of chromosomes in the population</th>
<th>Best value of S</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>2253</td>
</tr>
<tr>
<td>40</td>
<td>2353</td>
</tr>
<tr>
<td>60</td>
<td>2353</td>
</tr>
</tbody>
</table>

Table 3. Experiments of OPTPROJECT changing \( P_m \) (Project 1)

<table>
<thead>
<tr>
<th>( P_m )</th>
<th>Best value of S</th>
<th>Generation, which achieves ( S_{max} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.06</td>
<td>2353</td>
<td>7</td>
</tr>
<tr>
<td>0.07</td>
<td>2353</td>
<td>3</td>
</tr>
<tr>
<td>0.08</td>
<td>2353</td>
<td>6</td>
</tr>
<tr>
<td>0.1</td>
<td>2353</td>
<td>6</td>
</tr>
</tbody>
</table>

Fig. 11. Obtained optimal sequence of the Project 1 tasks

Similar tests are conducted for a larger schedule (Project 2) represented by the schedule in MS Project (Fig.12).

Table 4. Experiments of OPTPROJECT changing the number of generations (Project 2)

<table>
<thead>
<tr>
<th>Number of generations</th>
<th>Best value of S</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>39056</td>
</tr>
<tr>
<td>20</td>
<td>39274</td>
</tr>
<tr>
<td>40</td>
<td>39274</td>
</tr>
</tbody>
</table>

Table 5. Experiments of OPTPROJECT changing the chromosomes in the population (Project 2)

<table>
<thead>
<tr>
<th>Number of the chromosomes in the population</th>
<th>Best value of S</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>39200</td>
</tr>
<tr>
<td>80</td>
<td>39246</td>
</tr>
<tr>
<td>100</td>
<td>39248</td>
</tr>
</tbody>
</table>

Table 6. Experiments of OPTPROJECT Changing \( P_m \) (Project 2)

<table>
<thead>
<tr>
<th>( P_m )</th>
<th>Best value of S</th>
<th>Generation, which achieves ( S_{max} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.06</td>
<td>39406</td>
<td>18</td>
</tr>
<tr>
<td>0.08</td>
<td>39406</td>
<td>16</td>
</tr>
<tr>
<td>0.1</td>
<td>39406</td>
<td>10</td>
</tr>
<tr>
<td>0.5</td>
<td>39406</td>
<td>12</td>
</tr>
</tbody>
</table>

Fig. 12. Gantt chart of Project 2 activities
Conclusions and future work

The developed application OPTPROJECT provides a convenient graphical interface for managing projects. It allows the user to determine the parameters of the execution of the genetic algorithm depending on his needs. The parameters are: number of chromosomes in a population; number of generations and probability of mutation.

Output information derived from the application is presented in an appropriate graphical form. The application can be used to manage new projects in XML format. The user can view detailed results and use them for further analysis of the genetic algorithm.

The main advantage of the developed application OPTPROJECT is the possibility of implementing projects of each area of activity which consist of sequential tasks and the high costs have to be carried out at the end of the project. It can be used to manage both small and large projects. As with every application, there are also potential improvements in one or another direction. The future work is related with ability to directly record each resultant graph in Excel for subsequent processing of the results; ability to directly record the results in Word and more detailed Help for work with the application.

References: