Management of Software Projects with GAs

Enrique Alba J. Francisco Chicano*

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga
E.T.S. Ingeniería Informática, 29071 Málaga, Spain
{eat,chicano}@lcc.uma.es

1 Introduction

A Project Scheduling Problem consists in deciding "who does what" during the software project lifetime. This is a capital application in the practice of software engineering, since the total budget and human resources involved must be managed optimally to end in a successful project. In short, the main interest of a company is to reduce the time and the cost of the project, and these two goals are in conflict. The problem addressed in this article accounts for times, human skills, budget, and global complexity. This makes our study very realistic and similar to an actual software project planning scenario [3]. In this work we tackle the problem by using genetic algorithms to solve 36 different software project scenarios. Thanks to our newly developed instance generator we can perform structured studies about the influence in the solutions of the most important attributes of the problem. Our conclusions show that GAs are quite flexible and accurate techniques for this application, and can be an important tool for project managers [1]. Metaheuristics and, in particular, GAs are not as intensively applied in the software engineering domain as they are in fields like engineering, mathematics, economy, telecommunications, or bioinformatics [2]. However, the work of Clarke *et al.* [4] is a good reference for solving software engineering problems with metaheuristics.

The article is organized as follows. In Section 2 the Project Scheduling Problem is defined. Section 3 discusses the representation of the individuals and the fitness function, two very important issues in applying GAs to a problem. Finally, the experiments and results are explained in Section 4, and some conclusions and future work are discussed in Section 5.

2 The Project Scheduling Problem (PSP)

In the PSP the resources are people with a set of skills, a salary, and a maximum dedication to the project. We have E people, each one with a set of skills (from a subset of S skills considered

*Work partially funded by the spanish MCYT and FEDER under contract TIC2002-04498-C05-02. F. Chicano is supported by a grant from the Junta de Andalucía.

for the project), a monthly salary, and a maximum dedication to the project. The last is the ratio between the amount of hours dedicated to the project and the full working day length of the employee. The project has T tasks with a set of required skills and an effort (expressed in person per month) associated to each one. The tasks must be performed according to a Task Precedence Graph (TPG). The TPG indicates what tasks must be completed before the beginning of a new task. Our objectives are to minimize the cost p_{co} and the duration p_{du} of the project. The constraints are that each task must be performed by at least one person, the set of required skills of a task must be included in the union of the skills of the employees performing the task, and the total overtime p_{ov} must be zero. The number of skills measures the degree of specialization of the knowledge. That is, having a project with a globally larger number of skills the knowledge needed to perform the whole software project is divided into a larger number of portions than if a reduced number of skills were needed.

Once we know the elements of a problem instance, let us proceed to describe the elements of a solution to the problem. A solution can be represented with a matrix $\mathbf{X} = (x_{ij})$ of size $E \times T$. The element x_{ij} is the dedication degree of the ith employee to the jth task. For instance, if the employee performs the task with a 0.5 dedication degree he/she spends half of his/her working day in the task. This information helps to compute the duration of each task and, indeed, the start and end time of each one, i.e., the time schedule of the tasks. From this schedule we can compute the duration of the project. The cost can be calculated after the duration of the tasks accounting also for the dedication and salary of the employees. Finally, the overtime of each employee can be calculated using the time schedule of the tasks and the dedication matrix $\mathbf{X}$.

The PSP is related to the Resource-Constrained Project Scheduling (RCPS), an existing problem very popular in the literature that has been solved with exact techniques [5, 8, 10] and metaheuristic ones [6, 7, 9]. However, there are some differences between PSP and RCPS. First, in PSP there exists a cost associated with the employees and a project cost that must be minimized (in addition to the project duration). Besides, in RCPS there are several kinds of resource while PSP has one only kind of resource (the employee) with several possible skills. We must notice that PSP skills are different from RCPS resource kinds. A final difference is that each activity in the RCPS requires different quantities of each resource while PSP skills are not quantifiable entities.

3 Representation and Fitness Function

In this article we consider that no employee works overtime, so the maximum dedication of all the employees is 1. For this reason, the maximum value for x_{ij} is 1 and we have $x_{ij} \in [0, 1]$. On the other hand, we use a standard GA with binary string chromosomes to represent problem solutions and hence we need to discretize the interval $[0, 1]$ in order to represent the dedication degree x_{ij}. We distinguish eight values in this interval that are uniformly distributed, and employ three bits to represent them. The matrix $\mathbf{X}$ is stored into the chromosome in row major order. The chromosome length is $E \cdot T \cdot 3$. To compute the fitness of a solution $\vec{x}$ we use the next expression:

$$f(\vec{x}) = \begin{cases} 1/q & \text{if the solution is feasible} \\ 1/(q+p) & \text{otherwise} \end{cases} \quad (1)$$

where $q = w_{co} \cdot p_{co} + w_{du} \cdot p_{du}$ and $p = w_p + w_{nt} \cdot nt + w_{rs} \cdot rs + w_{ov} \cdot p_{ov}$

The fitness function has two terms: the quality of the solution (q) and the penalty for infeasible solutions (p). The two terms appear in the denominator because the goal is to minimize them, i.e., maximize $f(\vec{x})$. The quality term is the weighted sum of the project cost and duration. In this term, w_{co} and w_{du} are values weighting the relative importance of the two objectives. These weights allow us to tune the fitness according to our interest as project managers would probably do. In the penalization term p we weight and sum the parameters of the solution that make it infeasible, that is: the overtime of the project (p_{ov}), the number of tasks with no employee associated (nt), and the number of skills still required to perform all the tasks of the project (rs). Each of these parameters are weighted and added to the penalty constant w_p. This constant is included in order to separate the fitness range of the feasible solutions from that of the infeasible ones. The values for the weights used in this work are shown in Table 1. They have been obtained by exploring several solutions and with the aim of maintaining all the terms of the sum in the same range.

Table 1: Weights of the fitness function.

Weight	w_{co}	w_{du}	w_p	w_{nt}	w_{rs}	w_{ov}
Value	10^{-6}	0.1	100	10	10	0.1

4 Experiments and Results

In order to perform a meaningful study we must analyze several instances of the scheduling problem instead of focusing on just one, otherwise the conclusions could be biased. To do this we have developed an instance generator that creates new instances after setting a set of parameters such as the number of tasks, the number of employees, etc. A description of the generator, and the generator itself can be found at URL `http://tracer.lcc.uma.es/problems/psp`. For this study we generated 36 different instances and we solved them with a GA. We have separated the instances into two groups: the first one contains 18 instances where the number of skills demanded in the project (S) is fixed, and the skills per employee change between instances. The second one (also with 18 instances) has the complementary characteristic, i.e., the skills per employee are fixed and the total number of skills of the whole project changes.

The first benchmark have been generated by assigning different values to the following parameters: number of employees, number of tasks, and number of skills owned by the employees. The number of skills of the instances is always 10. The number of employees can be 5, 10, or 15 and the number of tasks 10, 20, or 30. Two ranges of values are considered separately for the number of skills of the employees: from 4 to 5, or from 6 to 7. With these values we want to represent the scenario of a medium or small software company that tackles different sized software projects. The skills per employee represent the training level of the personnel (higher training level means a larger number of skills). To solve the instances we use a genetic algorithm with a population of 64 individuals, binary tournament selection, 2-D single point crossover, bit-flip mutation, and elitist replacement of the worst (steady-sate genetic algorithm). The stop criterion is to reach 5000 steps of the main loop (5064 evaluations). We performed 100 independent runs for each instance and show in Table 2 left the number of runs in which a feasible solution was found.

Table 2: Percentage of runs that got a feasible solution in the two benchmarks.

First Bench.	*4-5 skills*			*6-7 skills*		
	employees			**employees**		
tasks	5	10	15	5	10	15
10	69	100	96	86	88	100
20	0	0	2	0	0	8
30	0	0	0	0	0	0

Second Bench.	*5 skills*			*10 skills*		
	employees			**employees**		
tasks	5	10	15	5	10	15
10	85	83	85	78	91	99
20	0	3	18	0	0	28
30	0	0	0	0	0	0

From the results we can notice that the instances with a larger number of tasks are more difficult than those with a smaller set of tasks. In the second row of results we also observe the inverse relationship between number of employees and difficulty. To better illustrate the meaning of these results, we plot the solutions obtained in a graph showing their cost and duration in two orthogonal axes (Figure 1). Cost and duration are clear trade-off criteria in any project. This is a kind of graph that a manager would like to see before taking any decision on the project. To indicate the solutions belonging to an instance we have put a label of the form `<tasks>-<employees>` near the solutions.

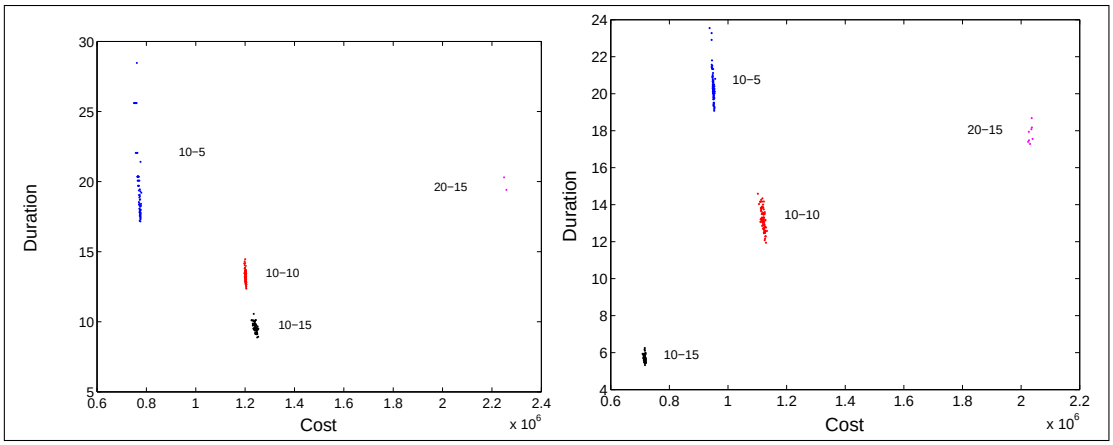

Figure 1: Results of the first benchmark with 4-5 (left) and 6-7 (right) skills per employee. Labels show the number of tasks and employees of the instance.

In the graphs, the solutions of each instance are observed as point swarms. Their elongated form depends on the scale of the axes (chosen to maintain the solutions of all the instances in the same graph). We can notice a slight inclination of the swarms showing the mentioned trade-off between cost and duration: when the cost of a solution is smaller its duration is higher. As we expected, when the number of employees decreases for a given number of tasks, the project spans longer in time. This result always holds despite each point swarm represents a different instance with different TPG. In the graphs we also noticed that the project is more expensive if a larger number of employees can be used in the solution, except for the instance with 10 tasks and 15 employees in Figure 1 right. A GA tool will help a manager to find such unexpected scenarios.

With the second benchmark we study the influence of the number of different skills in a project. This will be interesting for large companies where an assorted set of persons of varied experience are to be optimally assigned to software projects. For these experiments

we fix the range of the number of skills per task and employee from 2 to 3. The number of tasks can be 10, 20, or 30 and the number of employees takes values 5, 10, and 15, as in the previous instances. The number of different skills are 5 and 10. The parameters of the genetic algorithm are the same as in previous experiments, and we also performed 100 independent runs for every instance. In Table 2 right we show the results.

As in the previous benchmark we can see that an increment in the number of tasks means an increment in the difficulty of the problem. The participation of more employees usually implies a decrement of the difficulty of the instance (the project is easier to manage). However, we can state one new fact: we confirm that a larger number of demanded skills makes the instance easier (in general) to solve (see Table 2 right). After analyzing the solutions we find out that the tasks are performed by several employees with a low dedication degree and this seems to be beneficial for the cost and the duration of the project (compare the results of Figure 2 left and right).

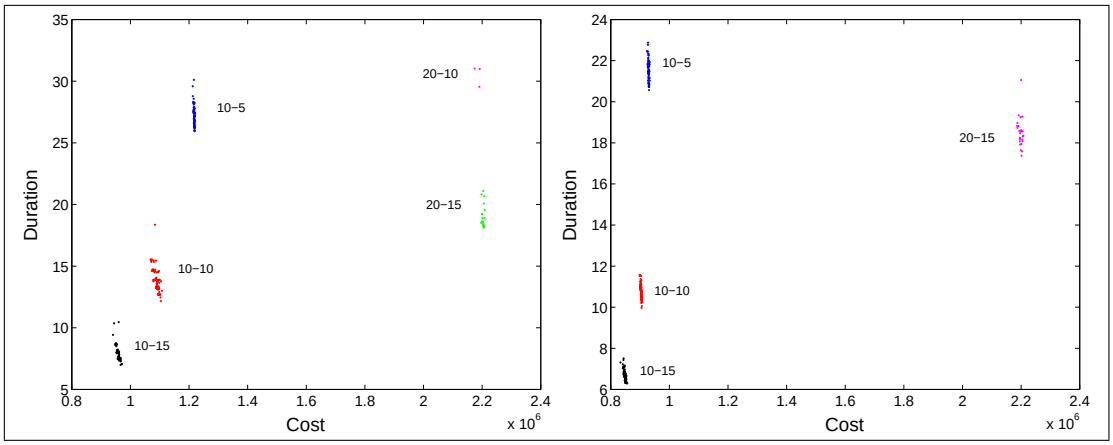

Figure 2: Results with 5 (left) and 10 (right) skills. Labels show the number of tasks and employees of the instance.

From Figure 2 we conclude that the cost of the project increases mainly with the number of tasks, and the duration of the project decreases with the increment in the number of employees. This was also observed in the previous instances. However, a solution with more employees reduces the overall cost of the project, a fact that was not observed before in knowledge-bounded companies (only well similar to 10-15 in Figure 1 right). The solutions of instances with a larger staff use all the employees at a low dedication to each task. In this way, the tasks are performed faster and the global cost of the project is low.

5 Conclusions

In this work we have tackled the general Project Scheduling Problem with genetic algorithms. A software manager can analyze different scenarios with such an automatic tool to take decisions on the best way to manage a project in a company. Furthermore, in our approach, the manager can adjust the fitness weights to better represent particular real world projects. The results show that the projects with more tasks are more difficult to solve, and their solutions are more expensive to implement in practice. In the same sense, the projects with a larger

number of employees at its disposal are easier to tackle and can be driven to a successful end in a shorter time. However, the relationship between employees and cost is not that simple: in some cases it is direct and in other cases it is inverse. This is where the GA can help. In the future we plan to add new instances with new features to study their influence on the difficulty of the instances, such as the complexity of dealing with a large team or the overhead of assigning a large set of tasks to an employee. In addition, we will solve the problem with other metaheuristic algorithms, maybe involving Pareto Dominance Concepts.

References

[1] Antoniol, Giuliano, Di Penta, Massimiliano, and Harman, Mark (2004): "A Robust Search-Based Approach to Project Management in the Presence of Abandonment, Rework, Error and Uncertainty". In: *Proceedings of the 10th International Symposium on Software Metrics (METRICS'04)*, September 2004, 172–183.

[2] Bäck, T., Fogel, D. B., and Michalewicz, Z. (1997): *Handbook of Evolutionary Computation*. Oxford University Press, New York, USA.

[3] Chang, Carl K., Christensen, Mark J., and Zhang, Tao (2001): "Genetic Algorithms for Project Management". In: *Annals of Software Engineering* **11**, 107–139.

[4] Clarke, J., Dolado, J. J., Harman, M., Hierons, R., Jones, B., Lumkin, B., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., and Shepperd, M. (2003): " Reformulating Software Engineering as a Search Problem". In: *Software Engineering. IEE Proceedings* **150** (3), 161–175.

[5] Demeulemeester, E., and Herroelen, W. (1992): "A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem". In: *Management Science* **38**, 1803–1818.

[6] Hindi, Khalil S., Yang, Hongbo, and Fleszar, Krzystof (2002): "An Evolutionary Algorithm for Resource-Constrained Project Scheduling". In: *IEEE Transactions on Evolutionary Computation* **6** (5), 512–518.

[7] Merkle, Daniel, Middendorf, Martin, and Schmeck, Hartmut (2002): "Ant Colony Optimization for Resource-Constrained Project Scheduling". In: *IEEE Transactions on Evolutionary Computation* **6** (4), 333–346.

[8] Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998): "An Exact Algorithm for Project Scheduling with Resource Constraints Based on a New Mathematical Formulation". In: *Management Science* **44** (5), 714–729.

[9] Palpant, Mireille, Artigues, Christian, and Michelon, Philippe (2004): "LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search". In: *Annals of Operations Research* **131**, 237–257.

[10] Talbot, B., and Patterson, J. (1978): "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems". In: *Management Science* **24**, 1163–1174.