Intelligent Software Project Scheduling and Team Staffing with Genetic Algorithms

Constantinos Stylianou[1] and Andreas S. Andreou[2]

[1]Department of Computer Science,
University of Cyprus,
75 Kallipoleos Avenue, P.O. Box 20537, Nicosia, 1678, Cyprus
cstylianou@cs.ucy.ac.cy
[2]Department of Electrical Engineering and Information Technology,
Cyprus University of Technology,
31 Archbishop Kyprianos Avenue, P.O. Box 50329, Limassol, 3603, Cyprus
andreas.andreou@cut.ac.cy

Abstract. Software development organisations are under heavy pressure to complete projects on time, within budget and with the appropriate level of quality, and many questions are asked when a project fails to meet any or all of these requirements. Over the years, much research effort has been spent to find ways to mitigate these failures, the reasons of which come from both within and outside the organisation's control. One possible risk of failure lies in human resource management and, since humans are the main asset of software organisations, getting the right team to do the job is critical. This paper proposes a procedure for software project managers to support their project scheduling and team staffing activities – two areas where human resources directly impact software development projects and management decisions – by adopting a genetic algorithm approach as an optimisation technique to help solve software project scheduling and team staffing problems.

Keywords: Software project management, project scheduling, team staffing, genetic algorithms

1 Introduction

A major problem that still exists in the area of software engineering is the high rate of software project failures. According to the 2009 Standish Group CHAOS Report [1], only 32% of software projects are delivered on time, within budget and with the required functionality, whereas 44% are delivered late, over budget and/or with less than the required functionality. The remaining 24% of software projects are cancelled prior to completion or delivered and never used. These figures reveal that project success rates have fallen from the group's previous study and, more alarmingly, that project failures are at the highest they've been in the last decade. Consequently, as more projects continue to fail, questions need to be asked of project management.

Questions such as “Was the team technically skilled to undertake the project?” or “Was it the project manager’s fault for under-costing and bad planning?” provide a strong incentive to address the shortcomings of software project management by focusing on two activities that project managers undertake. In particular, project scheduling and team staffing are examined because these activities are closely tied with human resources and since human resources are the only major resource of software development organisations, they are areas that warrant investigation. The paper approaches project scheduling and team staffing as an optimisation problem and employs a genetic algorithm to perform optimisation

The remainder of the paper is organised as follows: section 2 provides an overview of software project scheduling and team staffing activities. Subsequently, section 3 describes in brief how genetic algorithms work and presents how this optimisation technique was designed and adapted to solve the project scheduling and team staffing problem. Section 4 illustrates the experiments carried out on several test projects to evaluate the proposed approach followed by a discussion on the results obtained. Finally, section 5 concludes with a brief summary and notes on future work.

2 Literature Overview

2.1 Software Project Scheduling

One of the main responsibilities of a software project manager is to determine what work will be carried out, how and when it will be done. This responsibility consists of identifying the various products to be delivered, estimating the effort for each task to be undertaken, as well as constructing the project’s schedule. Due to the importance of this activity, it should have priority over all others, and furthermore, a project’s schedule needs to be updated regularly to coincide with the project’s current status.

One of the major issues of project scheduling is that of representation [2]. Specifically, the representations that most project scheduling tools provide “*cannot model the evolutionary and concurrent nature of software development*”. One of the most practical challenges project managers face in constructing project schedules is the fact that project scheduling problems are NP-complete. There are no algorithms that can provide an optimal solution in polynomial time, which means that brute force methods are basically inadequate [3, 4]. This problem is heightened due to the fact that software projects are intangible in nature and labour-intensive, and thus involve an even higher level of complexity and uncertainty. As a consequence of the uniqueness of software, project managers cannot completely depend on using experiences from previous projects nor can they use past project information unless used as indicative guidelines. Furthermore, goals may be different with respect to what type of optimisation is required (e.g., minimal cost, maximum resource usage, etc.). Finally, another issue is attributed to the level of information available to construct a project schedule. Similarly to software cost estimation, information available at the start of a project is usually missing or incomplete [2]. Therefore, any

mechanism adopted must be able to provide a means to continuously update the schedule [5].

2.2 Software Team Staffing

People and their role in teams are highly important for project success, and they are even taken into account in many cost estimation models, as for instance in the COCOMO model [6]. Therefore, employees in software development organisations should be characterised as "*human capital*", which is considered the most important asset of the company [7]. The more effectively this capital is managed, the higher the competitive benefit achieved over other organisations. Therefore, when a company undertakes the development of a new project, another of the main responsibilities of a project manager is to decide who will be working on the project. Team formation therefore is a task that, although seemingly easy at first, requires experience and careful execution. Not getting the right team to do the job could possibly lead to overrunning its schedule, exceeding its budget or compromising the necessary quality.

Recent changes in software development processes have also brought on changes into the way project managers view teamwork and this has lead to new approaches and methods being proposed for enhancing teamwork processes. An example of such attempt can be found in [8] who provide a Team Software Process (TSP) as an extension of the Personal Software Process (PSP). Another approach is proposed by [9] who employ a grey decision-making (fuzzy) approach that selects members based on their characteristics. The Analytical Hierarchy Process (AHP) is another method that can be used for team formation as a decision-making tool as shown in [10], who use multifunctional knowledge ratings and teamwork capability ratings. Recently, [11] have presented the Web Ontology Language (OWL) – a model that represents semantic knowledge – as a useful technique for team composition.

As stated in [12], the most common staffing methods available to software project managers rely heavily on the project manager's personal experiences and knowledge. However, these are highly biased techniques and subjectivity does not always yield the correct or best results. Another issue is the fact that because every project is unique, the application of a specific recruiting and staffing method on a project may not yield the expected results as it was applied on another project because of the differences in project characteristics [13]. And this links to the fact that skill-based and experience-based methods are not suitable enough for project managers to deal with interpersonal relationships and social aspects which strongly exist is software development organisations [14].

3 Methodology

3.1 Genetic Algorithm Overview

Project scheduling and team staffing may be considered optimisation problems, and as such will require specialised techniques to be solved. Genetic algorithms are one

such optimisation technique, with which it is possible to adequately model the mathematical nature of project scheduling and team staffing.

Genetic algorithms, introduced by John Holland in 1975 [15], work iteratively with populations of candidate solutions competing as a generation, in order to achieve the individual (or the set of individuals) considered as an optimal solution to a problem. Based on the process of natural evolution, their aim is for fitter individual solutions to prevail over those that are less strong at each generation. To achieve this, the fitness of every individual solution is evaluated using some criteria relative to the problem, and subsequently those evaluated highly are more probable to form the population of the next generation. Promoting healthy, better-off individuals and discarding less suitable, weaker individuals in a given generation is aided by the use of variations of the selection, crossover, and mutation operators, which are responsible for choosing the individuals of the next population and altering them to increase fitness as generations progress – making thus the whole process resemble the concept of 'survival of the fittest'.

3.2 Representation and Encoding

For the problem of project scheduling and team staffing, the candidate solutions for optimisation need to represent two pieces of information. On the one hand, schedule constraint information, regarding when and in which order tasks are executed and, on the other hand, skill constraint information, concerning the assignment of employees to tasks based on skill sets and experience required for a task. Fig. 1 below gives an example of the representation of a software project schedule containing 4 tasks and 5 possible employees. As shown, the genetic algorithm uses a mixed-type encoding: schedule information is represented by a positive, non-zero integer symbolising the start day of the task, whereas employee assignment information is represented by a binary code, wherein each bit signifies whether an employee is (a value of 1) or is not (a value of 0) assigned to execute the task.

1	10100	11	00010	16	01001	31	00110

Fig. 1. Example of project schedule representation

In Fig. 1, the first task starts at day 1 and employee 1 and 3 will execute it, task 2 starts at day 11 with only employee 4 assigned to it, and so on.

3.3 Fitness Evaluation Process

The evaluation of the fitness of each individual solution consists of an assessment across the two constraint dimensions that are the focus of this research (i.e., schedule constraints and skill constraints), and for each constraint dimension a corresponding objective function was constructed.

Schedule constraint objective. This objective concerns assessing the degree to which the dependencies of the tasks in the software project are upheld as well as the unnecessary delays existing between dependent tasks. The objective requires a maximisation function and requires information only from the start days of the tasks since it is not affected by which employees are assigned to carry out the task. For each $task_t$ in the project, the fitness is calculated based on Eq. 1. If a task's start day does not satisfy all its predecessor dependencies then it is given a value of zero. Otherwise, the number of idle days between the task and its predecessor task is calculated. The lower the number, the higher the value allocated. In the case where $task_t$ has more than one predecessor tasks, the predecessor task that ends the latest is used for the value $predecessor\_end\_day_t$.

$$f_{dependencies}(task_t) = \begin{cases} 0 & start\_day_t \leq predecessor\_end\_day_t \\ \frac{1}{1+idle\_days} & start\_day_t > predecessor\_end\_day_t \end{cases} \quad (1)$$

where

$$idle\_days = start\_day_t - predecessor\_end\_day_t - 1\ . \quad (2)$$

The values obtained for each task in the project are then averaged over the total number of tasks to give the final evaluation for this objective for the individual.

Skill constraint objective. Employees assigned to work on tasks are evaluated based on the degree of experience they possess in the skills required by the task. The objective function in Eq. 3 shows the formula used to make this evaluation. Experience in a skill is not considered as cumulative in this maximisation function and therefore is not presented as simply the summation of the skill experience value of all employees assigned. Instead, for each $skill_s$ required by a task, it makes use of the highest experience value (i.e., the level of the most experienced employee assigned with $skill_s$) and adds to that the mean level of experience of all the employees assigned to work on the task requiring $skill_s$.

$$f_{experience}(skill_s) = \max(experience\_levels_s) + \text{avg}(experience\_levels_s) \quad (3)$$

In this way, the objective function helps assign highly experienced employees to a task and simultaneously prevents the assignment of employees without the skills required (i.e., non-contributors) as the average experience of the team will be lowered. As with the previous objective function, the values obtained for each skill are then averaged over the total number of skills to produce an individual's final skill constraint evaluation.

Conflict objective. The two aforementioned objective functions individually target to realise the shortest project duration or to assign the most experienced employees respectively. However, when used together, there will be cases where conflicts will arise due to assigning one or more employees to work on tasks that have been scheduled to execute simultaneously. For this reason a third objective function was created to handle assignment conflicts by taking into account the number of days each

$employee_e$ has been assigned to work and how many of these days they have been assigned to more than one permitted task.

$$f_{conflict}\left(employee_e\right) = 1 - \frac{conflicting\_days_e}{total\_working\_days_e} \quad (4)$$

Subsequently, the overall fitness for conflicts is computed as the average of all employees. Finally, adding all three evaluations gives an individual's total fitness.

3.4 Parameters and Execution

The parameters selected for the genetic algorithm are summarised in Table 1 below.

Table 1. Genetic algorithm parameters

Population size: 100 individuals	Selection method: Roulette wheel
Maximum number of iterations: 10000	Crossover rate: 0.25
	Mutation rate: $2/chromosome\_length$

The genetic algorithm is initialised with random individuals and is set to execute for a maximum number of iterations. The population is evaluated using the objective functions described in subsection 3.3. However, it should be noted that each objective value is given a weight so as to allow guidance of the genetic algorithm based on the preference of project managers. For instance, a project manager may want to focus primarily on the construction of a project schedule with the shortest possible duration and secondarily on the experience of employees. In such a case, a higher weight will be assigned to the objective function evaluating the schedule constraint and a lower weight will be assigned to the objective function evaluating the skill constraint. If the most experienced employee is assigned to work on two parallel tasks (i.e., leading to a conflict), then preference will be given to keeping the duration of the project the same but assigning the next most experienced to either one of the tasks. Conversely, if a project manager prefers to have a team that is the most experienced and gives lower priority to project duration then, when a conflict occurs, the project schedule will grow in duration so that the most experienced employee remains assigned to the two tasks. Subsequent to the population evaluation, those individual solutions ranked the highest in terms of fitness will be passed on to the next generation after being transformed using crossover and mutation operators. This is repeated until a suitable solution (or number of suitable solutions) has been found.

4 Application and Experimental Results

4.1 Design of Experiments

Experiments were carried out to validate the approach on several aspects. Foremost, it is essential to examine whether the objective functions used in the optimisation algorithm were sufficient enough to produce the correct results individually but also

in their competitive environment. For this purpose, a number of test projects of different sizes were designed and constructed that also allowed investigation of the behaviour of the optimisation algorithm. The two test projects are depicted in the task precedence graph (TPG) in Fig. 2 below, which was used in [3]. The smaller test project comprises a subset of 10 tasks (T1-T10) of the TPG, whereas the larger test project contains all the tasks (T1-T15) in the TPG. All dependencies between tasks are finish-to-start, and the duration and the set of skills required for each task is given inside the respective tasks nodes. In addition, Table 2 provides the degree of experience that employees possess in the skills required by the project's tasks.

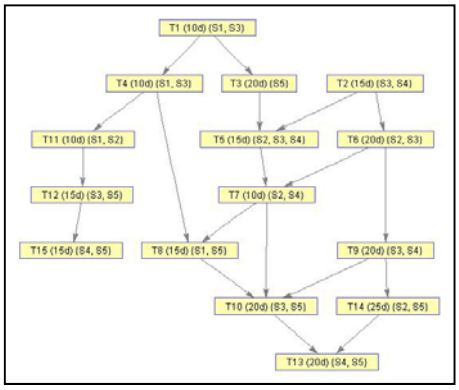

Fig. 2. Task precedence graph for test projects

Table 2. Employees' level of experience for the test projects

	S1	S2	S3	S4	S5
E1	0	0	0.4	0.8	0
E2	0.2	0	0.4	0	0
E3	0	0.8	0	0	0
E4	0	0	0.4	0.8	0.6
E5	0	0.6	0	0	0
E6	0	0.6	0.4	0.8	0
E7	0	0.4	0.4	0	0
E8	0	0.6	0.6	0.8	0
E9	0	0.2	0.4	0	0
E10	0.6	0.4	0	0	0.6

The genetic algorithm was executed thirty times for each test project in each experiment. The results reported in the following subsections contain examples of some of the best executions that converged or that came very close to converging to the optimal solution in each experiment. For Experiment 1, roughly 60% of the executions resulted in finding the optimal solution (i.e., the project schedule with the shortest makespan), whereas for Experiment 2 all executions obtained the optimal solution (i.e., the project schedule with the most experienced employees assigned). Finally, for Experiment 3, only around 5% of executions converged to the correct solution. It should also be noted here that genetic algorithms are random in nature with respect to initialisation and application of genetic operations. Because this randomness affects convergence, that is,. the number of iterations required to find the optimal solution, the time taken to complete an execution varied. Hence, any figures regarding execution time can only give a very rough indication of the overall behaviour of the genetic algorithm in terms of performance. Execution time ranged between 17 sec and 5 min, depending also on the size of the project.

4.2 Results and Discussion

Experiment 1. The first experiment involved executing the genetic algorithm on the test projects to evaluate the objective function for the schedule constraint only. This was done to assess that dependencies between tasks were in fact satisfied and no

unnecessary delays existed. An example of the behaviour of the genetic algorithm, one for each test project, can be seen in Fig. 3. For the smaller test project the optimal solution was found around 1000 iterations whereas, as expected, for the larger test project the optimal solution required a higher number of iterations (roughly 3000). Construction of the corresponding optimal project schedules correctly shows that the shortest possible duration for the first test project is 90 days and for the second test project the shortest makespan is 110 days (Fig. 4).

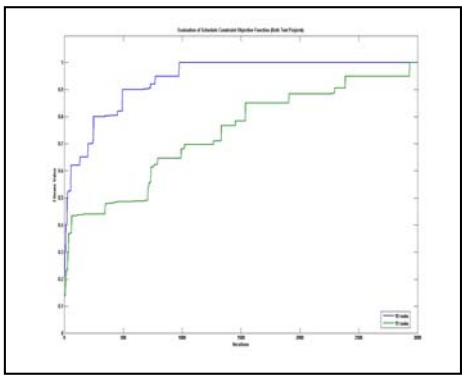

Fig. 3. Evolution of the best individual for the two test projects

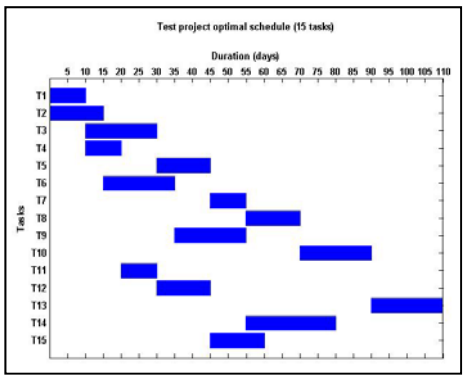

Fig. 4. Optimal project schedule for large test project (15 tasks)

Experiment 2. The second experiment examined whether the genetic algorithm was indeed able to find the optimal project team with regards to experience by only evaluating the objective function for the skill constraint. For both test projects, the genetic algorithm successfully managed to assign to each task the most experienced employee or group of employees so that all skills were satisfied and no "idle" or "surplus" employees were used. Table 3 shows an example of a resulting assignment matrix that displays which employees will staff the small software test project.

Table 3. Employee-task assignment matrix (small test project)

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
E4	0	0	1	0	0	0	0	0	0	1
E6	0	0	0	0	0	0	1	0	0	0
E8	1	1	0	1	1	1	0	0	1	0
E10	1	0	0	1	0	0	0	1	0	0

Experiment 3. The third and final experiment was carried out to investigate the behaviour of the genetic algorithm when all three objectives were included in the evaluation. Since the objective functions are considered to be competing with each other, each objective function was multiplied by its preference weight as explained in subsection 3.4.

Firstly in this series of experiments, greater preference was given to the schedule constraint than to the skill constraint. The results obtained showed that with all three

objective functions active, the genetic algorithm was successfully able to find the optimal schedule for both test projects and, in order to avoid conflicts, managed to assign the next best employees in terms of experience to parallel tasks. An example of the assigned employees of the smaller test project is given in Table 4.

Table 4. Employee-task assignment matrix keeping shortest possible project schedule

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
E2	1	0	0	1	0	0	0	0	0	0
E4	0	0	1	0	0	0	0	0	0	1
E6	0	0	0	0	1	0	1	0	0	0
E8	0	1	0	0	0	1	0	0	1	0
E10	1	0	0	1	0	0	0	1	0	0

Secondly, a higher preference was given to the skill constraint and a lower preference to the schedule constraint. This was done to examine whether the genetic algorithm was able to keep the most experienced employees assigned by lengthening the project duration. Results obtained here showed that the genetic algorithm found it difficult to reach an optimal solution. Specifically, runs carried out on the test projects show that, as expected, the genetic algorithm draws its attention to the skill constraint objective (as seen in the example in Fig. 5), and thus the project duration is prolonged (Fig. 6). However, a minor dependence violation in one task of the order of 1 day is caused due to the non-multiobjective nature of the algorithm.

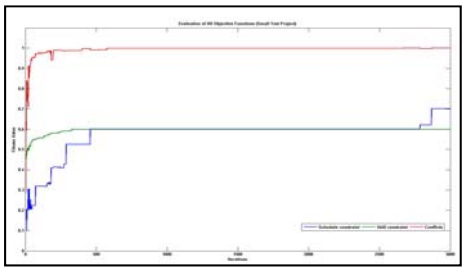

Fig. 5. Evolution of best individual for small test project (10 tasks)

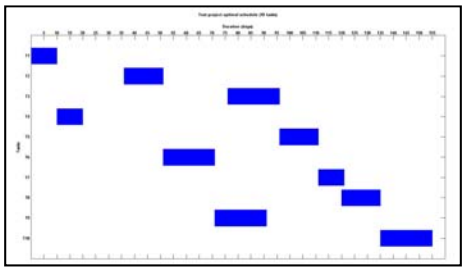

Fig. 6. Project schedule for small test project (10 tasks)

5 Concluding Remarks

This paper presented an approach to solving the problem of software project scheduling and team staffing by adopting a genetic algorithm as an optimisation technique in order to construct a project's optimal schedule and to assign the most experienced employees to tasks. The genetic algorithm uses corresponding objective functions to handle constraints and the results obtained when using either one of the objective functions show that the genetic algorithm is capable of finding optimal solutions for projects of varying sizes. However, when the objective functions were

combined, the genetic algorithm presents difficulties in reaching optimal solutions especially when having preference to assign the most experienced employees over the project's duration. Through observation of a number of executions, it was noticed that in this case the genetic algorithm couldn't reduce idle "gaps" or was not able to produce a conflict-free schedule. One possible reason for this observation is due to the competitive nature of the objective functions, and a definite improvement to the approach will be to use multi-objective optimisation rather than using the aggregation of individual objective functions, which is how the genetic algorithm presently works. This could possibly be a means to handle the competitiveness of the objective functions and also allow for a set of optimal solutions to be produced throughout both constraint dimensions simultaneously, thus removing the need for using weights to give preference to either one of the constraints. Also, refinements may be needed to help the algorithm escape from local optima and, thus, improve its convergence rate.

References

1. Standish Group.: Standish Group CHAOS Report. Standish Group International, Inc, Boston (2009)
2. Chang, C.K., Jiang, H., Di, Y., Zhu, D., Ge, Y.: Time-Line Based Model for Software Project Scheduling with Genetic Algorithms. Inform. Software Tech. 50(11), 1142--1154 (2008)
3. Chang, C.K., Christensen, M.J., Zhang, T.: Genetic Algorithms for Project Management. Ann. Softw. Eng. 11(1), 107--139 (2001)
4. Pan, N., Hsaio, P., Chen, K.: A Study of Project Scheduling Optimization using Tabu Search Algorithm. Eng. Appl. Artif. Intel. 21(7), 1101--1112 (2008)
5. Joslin, D., Poole, W.: Agent-based Simulation for Software Project Planning. In: 37th Winter Simulation Conference, pp. 1059--1066. IEEE Press, New York (2005)
6. Boehm, B.W.: Software Engineering Economics. Prentice Hall Inc., New Jersey (1981)
7. Acuña, S.T., Juristo, N., Moreno, A.M., Mon, A.: A Software Process Model Handbook for Incorporating People's Capabilities. Springer, New York (2005)
8. Humphrey, W.S.: The Team Software Process℠ (TSP℠). Technical Report, Carnegie-Mellon University (2000)
9. Tseng, T.-L., Huang, C.-C., Chu, H.-W., Gung, R.R.: Novel Approach to Multi-Functional Project Team Formation. Int. J. Proj. Manage. 22(2), 147--159 (2004)
10. Chen S.-J., Lin, L.: Modeling Team Member Characteristics for the Formation of a Multifunctional Team in Concurrent Engineering. IEEE T. Eng. Manage. 51(2), 111--124 (2004)
11. Chi, Y., Chen, C.: Project Teaming: Knowledge-Intensive Design for Composing Team Members. Expert Sys. Appl. 36(5), 9479--9487 (2009)
12. Acuña, S.T., Juristo, N., Moreno, A.M.: Emphasizing Human Capabilities in Software Development. IEEE Softw. 23(2), 94--101 (2006)
13. Wi, H., Oh, S., Mun, J., Jung, M.: A Team Formation Model Based on Knowledge and Collaboration. Expert Sys. Appl. 36(5), 9121--9134 (2009)
14. Amrit, C.: Coordination in Software Development: The Problem of Task Allocation. In: 27th International Conference on Software Engineering, pp. 1--7. ACM, New York (2005)
15. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Michigan (1975)