A Novel Multiobjective Formulation of the Robust Software Project Scheduling Problem

Francisco Chicano[1], Alejandro Cervantes[2],
Francisco Luna[1], and Gustavo Recio[2⋆]

[1] University of Málaga, Spain
{chicano,flv}@lcc.uma.es,
[2] University Carlos III of Madrid, Spain
{acervant,grecio}@inf.uc3m.es

Abstract. The Software Project Scheduling (SPS) problem refers to the distribution of tasks during a software project lifetime. Software development involves managing human resources and a total budget in an optimal way for a successful project which, in turn, demonstrates the importance of the SPS problem for software companies. This paper proposes a novel formulation for the SPS problem which takes into account actual issues such as the productivity of the employees at performing different tasks. The formulation also provides project managers with robust solutions arising from an analysis of the inaccuracies in task-cost estimations. An experimental study is presented which compares the resulting project plans and analyses the performance of four different well-know evolutionary algorithms over two sets of realistic instances representing the problem. Statistical parameters are also provided in order to help the project manager in the decision process.

Keywords: Software Project Scheduling, Robustness, Multi-objective Optimisation, Evolutionary Algorithms

1 Introduction

As software projects become larger, the need to control people and processes, and to efficiently allocate resources turn out to be increasingly important. Managing such projects usually involves scheduling, planning, and monitoring tasks. This paper focuses on minimising both, the project cost and its make-span, during the assignment of employees to particular tasks in the context of a software project. The problem studied here is known in the literature as the Software Project Scheduling (SPS) problem [2].

In general, the solution of a multi-objective problem, such as SPS, consists of a set of non-dominated solutions known as the *Pareto optimal set*, which is often called *Pareto border* or *Pareto front* when plotted in the objective space [3].

⋆ This work has been partially funded by the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04. It has also been partially funded by the Andalusian Government under contract P07-TIC-03044.

Solutions within this set are optimal in the sense that there are not solutions which are better with regards to one of the objectives without achieving a worse value in at least another one. Particularly, in the context of the SPS problem, it is not possible to reduce the project cost without increasing its make-span (or vice versa). Previous works in the literature have addressed the SPS problem using single-objective and multi-objective formulation with meta-heuristics [2]. The contribution of this research differs from previous works in three ways. First, a new formulation of the problem is presented which involves more realistic assumptions than the previous ones [2], e.g. the need of taking into account several constraints was removed simplifying then the optimisation process. Second, the concept of robustness of a solution was introduced into the formulation of the problem in order to deal with inaccuracies in task-cost estimations. Third, the experimental study carried out here consists on applying four multi-objective evolutionary algorithms to the problem using three different robustness approaches over two different instances derived from a realistic software project. The resulting solutions were analysed using correlation measures between solution features and objective function values.

This paper is organised as follows. The new formulation of the SPS problem and the way robustness is considered are described in Section 2. The experimentation carried out with the corresponding analysis of results is detailed in Section 3. Finally, Section 4 deals with discussion of the main findings and contributions of this research.

2 The SPS Problem

Consider the set of people potentially involved in a software project E, where each person is denoted as $e_i \in E$, with i varying from 1 to $|E|$ (the number of employees), and e_i^s being the salary of an employee. The set of tasks to be performed in the project and each individual task are referred to as T and $t_j \in T$ respectively (with j varying between 1 and $|T|$). The cost in person-hour of task t_j is denoted with t_j^c. The tasks must be performed according to a Task Precedence Graph (TPG) that indicates which tasks must be completed before a new task is started. The TPG is an acyclic directed graph $G(T, A)$ which nodes represent the tasks and an arc $(t_i, t_j) \in A$ exists if task t_i must be completed, with no other intervening tasks, before task t_j can start. Each instance of the problem includes a productivity matrix $\boldsymbol{P}$ of size $|E| \times |T|$ in which element $\boldsymbol{P}_{i,j} \in [0, 1]$ is a positive real value which describes the productivity of employee e_i in task t_j. This productivity value is related to the time required by the employee to finalise the task. If employee e_i is working alone in task t_j then $t_j^c / \boldsymbol{P}_{i,j}$ hours are required to complete the task.

A solution to this problem $x = (d, r, \boldsymbol{q})$ consists of a real valued vector of employee dedication $d \in \mathbb{R}^{|E|}$, an integer valued vector of task delays $r \in \mathbb{N}^{|T|}$ and an integer valued matrix of priorities $\boldsymbol{q} \in \mathbb{N}^{|E| \times |T|}$. Each component d_i of the dedication vector refers to the percentage of a full working day that employee e_i spends in tasks related to the project. Thus, $d_i = 0.5$ means that half working day is spent in the project by employee e_i. If working alone in task t_j with productivity $\boldsymbol{P}_{i,j} = 1$ then the task takes $2t_j^c$ hours for completion. The

component r_j within the vector of task delays refers to the number of hours that task t_j is delayed with respect to the earliest possible starting time, e.g. if task t_j can start at time h, then, applying task delays it will start at time $h + r_j$. Task delays where introduced in the formulation as under certain circumstances they are needed in order to represent optimal solutions in terms of make-span. Without considering delays, the model can only generate solutions with as many tasks as possible processed in parallel, i.e. tasks are started as soon as the TPG allows it. If some of those paralleled tasks are in the critical path (that is, their make-span highly influences the total make-span), a better total make-span can be achieved if the critical tasks are completed as soon as possible. The use of task delays allow the model to represent such solutions: non-critical tasks can start later than allowed by the TPG, so they have less parallelisation than critical tasks. Critical tasks get as much dedication as possible minimising their contribution to the total make-span. The priority matrix $\boldsymbol{q}$ specifies which task is performed by each employee. In the case in which an employee is simultaneously working in several tasks, the matrix also specifies the distribution of employee time between parallel tasks. An employee e_i works in task t_j when $\boldsymbol{q}_{i,j} > 0$ and $\boldsymbol{P}_{i,j} > 0$. If employee e_i is working at a given time τ in tasks t_{j_1}, t_{j_2}, ..., t_{j_l} then the amount of time dedicated to task t_{j_m} is given by $\boldsymbol{q}_{i,j_m}/(\sum_{k=1}^{l} \boldsymbol{q}_{i,j_k})$.

Optimising the cost and the make-span of the proposed scheduling of the software project is the aim behind this research. Therefore, the evaluation of cost and make-span becomes highly important. Consider a discrete time where the time variable is represented by τ. The working hours in the software company in which the project is begin developed are represented as finite values of the time variable $\tau = 0, 1, 2, \ldots$ (being $\tau = 0$ the starting time of the project).

Since the employee dedication to a task is time dependent (due to simultaneous tasks), computing the make-span of the project involves an auxiliary time-dependent real valued vector of manpower for each task. Such vector will be denoted as π, where π_j refers to the manpower of the team at performing task t_j, e.g. if $\pi_j(7) = 2$ then at time $\tau = 7$ the remaining cost of task t_j is reduced in 2 persons-hour. The set of finalised and active tasks at time τ based on the values of $\pi(\tau')$ for $\tau' < \tau$ are defined as

$$done(\tau) = \left\{ t_j \in T \middle| \sum_{\tau'=0}^{\tau-1} \pi_j(\tau') \geq t_j^c \right\}, \tag{1}$$

$$active(\tau) = \{t_j \in T | \forall t_i, (t_i, t_j) \in A : t_i \in done(\tau - r_j)\} - done(\tau), \tag{2}$$

where A stands for the arc set of the TPG. Notice that the computation of both, the $active(\tau)$ and $done(\tau)$ sets only depends on the values of $\pi(\tau')$ for $\tau' < \tau$. In particular, $done(0)$ and $active(0)$ do not depend on π, as $done(0) = \emptyset$ and $active(0)$ is the set of initial tasks: $active(0) = \{t_j \in T | \nexists t_i \in T, (t_i, t_j) \in A\}$.

The vector $\pi(\tau)$ can be computed for each time step $\tau = 0, 1, \ldots$ in an iterative manner as follows:

$$\pi_j(\tau) = \begin{cases} \sum\limits_{e_i \in E : \boldsymbol{q}_{i,j} > 0} \frac{d_i \cdot \boldsymbol{P}_{i,j} \cdot \boldsymbol{q}_{i,j}}{\sum_{t_k \in active(\tau)} \boldsymbol{q}_{i,k}} & \text{if } t_j \in active(\tau), \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The project make-span is: $makespan(x) = \min\{\tau \in \mathbb{N} | done(\tau) = T\}$, that is, the *make-span* refers to the amount of time required to complete all the tasks in the project. A well defined make-span involves that all tasks must be performed by at least one employee with non-zero productivity in the corresponding task. This is the only constraint imposed to the solutions.

The cost is computed by multiplying the salary per hour of each employee, the dedication of the employee and the number of hours dedicated to tasks in the project. Then the salaries for all employees are sum together to compute the total cost of the project, that is:

$$cost(x) = \sum_{e_i \in E} e_i^s \cdot d_i \cdot |\{\tau \in \mathbb{N} | \exists t_j \in active(\tau) : \boldsymbol{q}_{i,j} \cdot \boldsymbol{P}_{i,j} > 0\}| \,. \tag{4}$$

Considering the expressions for the makespan and the cost, the SPS problem can be modelled as a bi-objective optimisation problem with objective function $f(x) = (cost(x), makespan(x))$.

2.1 Adding Robustness to the Solutions

In a real scenario, task costs are usually estimations made on the basis of previous experiences and they are not accurate. Indeed, a review of studies in estimation accuracy points out that software projects overspend on average 30-40% more effort than estimated [10]. Taking into account these uncertainties in the problem statement allows search algorithms to propose not only good solutions according to the main objectives (cost and make-span), but also to provide robust solutions whose cost and make-span are not sensitive to changes in the the cost of individual tasks due to inaccuracies of the initial estimations. These disturbances in task costs can be modelled by using a multivariate random variable $\mathcal{T}^c = \{t_1^c, \ldots, t_{|T|}^c\}$ following a probability distribution $\mathcal{C}$. Given a solution x, the project cost and make-span are now defined by a bivariate random variable S that can be computed as $S(x) = (makespan(x), cost(x)) = f(x, \mathcal{T}^c)$ where $\mathcal{T}^c$ was explicitly introduced in the notation to clarify that the solution evaluation depends on the varying estimated cost of the tasks (represented here with its random variable).

The average and the standard deviation of each component of $S(x)$ are used as a measure of the quality and the robustness of each objective for a given solution, respectively. These values are computed by sampling over a number of H simulations of $\mathcal{T}^c$. The bi-objective formulation of the problem is, thus, transformed into a four-objective one:

$$f(x) = (makespan_{avg}(x), makespan_{sd}(x), cost_{avg}(x), cost_{sd}(x)), \tag{5}$$

where sub-indices in the original objective functions were used to denote the average and the standard deviation of the sampling performed to compute the robustness. As before, these four objectives are expected to be minimised.

Three different robustness scenarios are being considered. The first one, denoted as NR, assumes perfect knowledge on the task cost. The second one assumes that only one task has been miss-estimated (noted as OTR). The third

one assumes that all tasks could have been miss-estimated (noted as STR). The probability distribution used to generate the perturbation in the cost of a task in the two last scenarios is such as that each t_j^c is multiplied by a value uniformly drawn from the interval $[0.5, 2.0]$. That is, each task can be carried out from half to double of its original estimated cost.

2.2 Comparison against other Scheduling Problems

An analogy can be established between the shop scheduling [6] and SPS problems. The tasks would be the same in both problems, employees in SPS would be analogous to machines in shop scheduling problems, the productivity $\boldsymbol{P}_{i,j}$ of employee e_i in task t_j when considering SPS would be related to the length of task j in machine i for shop scheduling problems. However, in shop scheduling only one machine can perform a task, while in SPS problems tasks are performed by working teams of employees. In addition, the decision variables in SPS determine the dedication of an employee to a software project whereas in the case of shop scheduling the "dedication" or efficiency of a machine cannot be modified.

Another problem related to SPS is the Resource-Constrained Project Scheduling (RCPS) [12]. In RCPS there are several kinds of resources while SPS involves only one: the human resource. Each activity in RCPS requires different amounts of each resource while SPS does not imposes a minimum or maximum number of employees in a working team developing a task.

Two important works that also use modelling of software project scheduling were presented in Gutjahr *et al.* [7] and Chang *et al.* [1]. The former includes a model of learning capabilities of employees and a portfolio selection. In the second, a solution accounts for the time variation of the assignment of employees to tasks. The drawback of complex formulations, like the previous ones, is the large number of parameters that the project manager must configure to provide a complete instance of the problem, which in turn increases the chances of miss-estimating such parameters. Hence, the improved accuracy obtained using a more realistic formulation of the problem turns out to be limited by a larger inaccuracy of the problem instance parameters. Then, a new formulation which is a trade-off between realistic (but complex) and simple (and unrealistic) formulations is proposed in this work.

3 Experimental Study

Four meta-heuristics have been used to carry out the experimental study: NSGA-II [5], SPEA2 [14], PAES [9], and MOCell [11]. They all use evolutionary computation which is by far the most popular meta-heuristic technique for solving MOPs due to their ability in finding a set of trade-off solutions in one single run [3, 4]. Binary tournament selection, two-point crossover and a random mutation that randomly chooses a value within the range defined for each variable were used in NSGA-II, SPEA2 and MOCell. On the other hand, the PAES execution sequence consisted of a single individual population which is iteratively modified by using only random mutation (no crossover operator was used), pareto front solutions in this case are obtained by using an external archive where all non-dominated solutions are stored. Population sizes of 100 individuals were used for

NSGA-II, SPEA2, and MOCell, whereas the pareto front size was limited to 100 solutions in the four approaches. Crossover and mutation rates were $p_c = 0.9$ and $p_m = 1/L$ respectively, where L refers to the length of the tentative solution. Aiming at performing a fair comparison between different algorithms, the stopping criterion for them all consisted in computing 1000000 function evaluations. Finally, the size of the Monte Carlo sampling used to evaluate the solutions of the robust SPS versions was set to a neighbourhood of $H = 100$.

Two quality indicators have been used to measure the performance of the multi-objective algorithms: the hyper-volume (HV) [15] and the attainment surfaces [8]. The HV is considered as one of the more suitable indicators by the EMO community since it provides a measure that takes into account both the convergence and diversity of the obtained approximation set. The empirical attainment surfaces have been defined to be a kind of "average" Pareto front of a randomised multi-objective algorithm. For each pair of algorithm vs test problem instance, 100 independent runs were carried out. The HV indicator and the attainment surfaces were then computed. In the case of HV computations, a multiple comparison test was carried out in order to check if the differences were statistically significant or not. All the statistical tests were performed with a confidence level of 95%.

Two realistic instances that are variations of a project scheduling which is available at the online repository of the MS Project tool will be solved in this research. The same TPG (see Fig. 1), tasks cost and number of employees as in the original instance will be used and the values for the employees salary and the productivity matrix will also be provided. Table 1 summarises the above information.

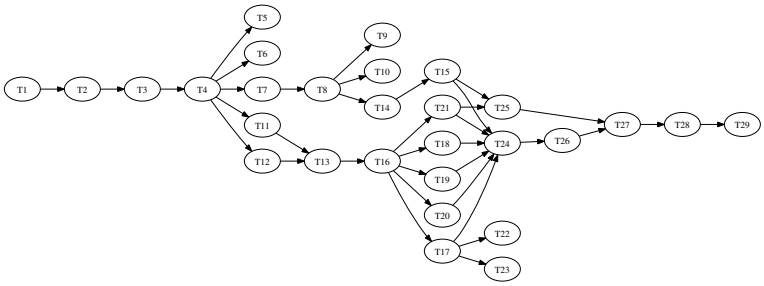

Fig. 1. Task Precedence Graph for the two instances of the SPS problem being solved.

Both instances, denoted with `ms1` and `ms2`, differ in the definition of their productivity matrix. In `ms1` all the values in the productivity matrix are 0 or 1 and are based on the original assignment of employees to tasks in the sample project (denoted as "base solution"). On the other hand, instance `ms2` contains a more flexible resource productivity table, with larger overlap between resources' abilities, and also fractional (not 1.0) productivity in tasks.

3.1 Performance of the Algorithms

A comparison of the performance of the four multi-objective algorithms within the three robustness scenarios is carried out in this Section. The performances

Table 1. Productivity matrices $\boldsymbol{P}_{i,j}$, task cost t_j^c and employee salary e_i^s.

Emp.			Task (t_j)																											
e_i	e_i^s		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
e_1	50	ms1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		ms2	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e_2	40	ms1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1
		ms2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	.5	0	0	1	1	
e_3	10	ms1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0
		ms2	0	0	0	0	0	0	0	.3	.3	.3	0	0	.5	0	0	0	0	.5	0	.5	0	0	0	.5	0	.5	0	0	0
e_4	15	ms1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	0
		ms2	0	0	0	0	0	0	0	1	1	1	.5	.5	.5	0	0	0	0	0	.8	0	.8	0	0	.8	.8	.8	.8	.8	0
e_5	20	ms1	0	1	1	1	1	1	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	0
		ms2	0	.5	.5	.5	.5	.5	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	0
e_6	30	ms1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0
		ms2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	.8	0	0	.8	0
e_7	30	ms1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0
		ms2	0	.7	.7	.7	.7	.7	.7	.7	.7	.7	.7	.7	.7	.7	.7	0	0	0	0	0	0	0	0	1	1	1	1	1	0
t_j^c			*6*	*680*	*408*	*8*	*10*	*10*	*378*	*10*	*10*	*10*	*162*	*48.6*	*8.8*	*720*	*6*	*198*	*180*	*6*	*108*	*6*	*30*	*36*	*36*	*18*	*540*	*120*	*180*	*450*	*3*

have been evaluated using the HV indicator which values are summarised in Table 2. The best performances are highlighted in a dark grey background whereas second to best are shown in light grey. We also mark with * the results having statistically significant differences with the best result. Several conclusions can be drawn from these values. Both NSGA-II and MOCell obtained the best (largest) values for the two instances (as well as many of the second to best values). NSGA-II resulted in the best performance when tackling the robust versions of the instances (in 3 out of the 4 scenarios the approximated Pareto front with best HV indicator was returned). On the other hand, MOCell seems to be specially well suited for the non-robust setting, yielding the higher HV indicator for the two instances. PAES seems to be clearly the worst algorithm with respect to this indicator, specially for the robust versions. The uncertainty in the objective functions could be the main reason behind this fact. Regarding the runtime, all the algorithms require between 2.5 and 5 minutes in the NR scenario, while they require around 5 hours in the OTR and STR scenarios.

Table 2. Median and IQR of the HV value for the two instances.

	NSGAII	SPEA2	PAES	MOCell	NSGAII	SPEA2	PAES	MOCell
Rob.	`ms1`				`ms2`			
NR	$0.943^{*}_{0.000}$	$0.943^{*}_{0.000}$	$0.518^{*}_{0.065}$	$0.944_{0.000}$	$0.904^{*}_{\pm 0.000}$	$0.905^{*}_{\pm 0.001}$	$0.543^{*}_{\pm 0.031}$	$0.905_{\pm 0.000}$
OTR	$0.829^{*}_{0.027}$	$0.807^{*}_{0.030}$	$0.328^{*}_{0.039}$	$0.816_{0.032}$	$0.738_{\pm 0.025}$	$0.730_{\pm 0.018}$	$0.287^{*}_{\pm 0.020}$	$0.695^{*}_{\pm 0.043}$
STR	$0.746_{0.028}$	$0.688^{*}_{0.063}$	$0.345^{*}_{0.036}$	$0.742_{0.025}$	$0.764_{\pm 0.025}$	$0.717^{*}_{\pm 0.030}$	$0.387^{*}_{\pm 0.032}$	$0.769_{\pm 0.022}$

3.2 Analysis of solutions

This section focuses on analysing the solutions obtained using the multi-objective algorithms. Figure 2 (left) shows the result of an NSGA-II execution over both instances using the NR approach. The base solution for instance `ms1` is close to a minimum-make-span solution, as all available employees are committed to tasks for which they have non-zero productivities. None the less, the algorithm is able to improve this minimum make-span. The Pareto front includes solutions with smaller cost which were obtained by reducing the dedication of the most expensive resources when developing their tasks. In instance `ms2` improvements in both, cost and make-span, using NSGA-II with respect to the base solution were also observed.

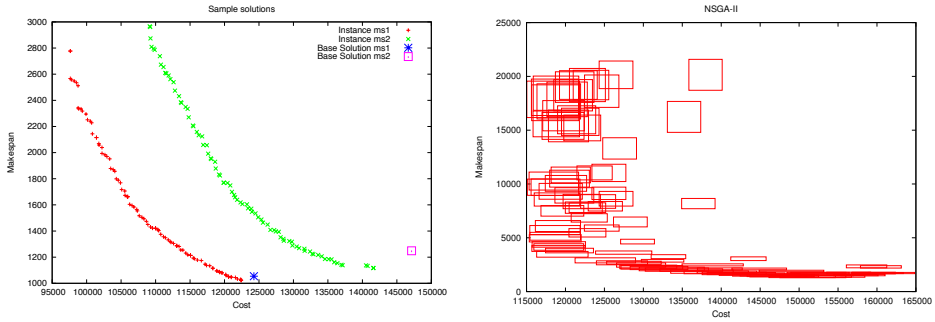

Fig. 2. Pareto front sample and base solution for the two instances (left). 50%-attainment surface for `ms1` in the STR robust approach (right). The position of the boxes is determined by the average value and the size by the standard deviation.

Fig. 2 (right) shows the 50%-attainment surface of NSGA-II for `ms1` within the STR scenario. A four objective problem requires 4D data to be represented in order to visually inspect the resulting Pareto fronts. In order to show both the quality (average) and the robustness (standard deviation) in the cost and make-span of a project scheduling problem, the approach taken consists on displaying boxes such that the position of the center of a box is defined by the two average values ($cost_{avg}(x)$ and $makespan_{avg}(x)$), whereas the width and the height are proportional to $cost_{sd}(x)$ and $makespan_{sd}(x)$. It is worth mentioning that when the average values of cost and make-span are reduced (bottom left corner of the plot), the standard deviation is increased (larger boxes). It was also observed that high-cost solutions show a low make-span and are quite robust in make-span, whereas low-cost solutions are not robust in make-span or cost. This can be explained by the larger need of average parallelism required by low make-spam solutions, thus, task deviations are distributed among several employees working in the same task.

Consider now the features of the solutions x in the approximated Pareto front. In particular, a detail analysis must be done accounting for the number of employees performing each task t_j (denoted as $t_j^e(x)$) and the average number of tasks that each employee e_i performs in parallel (denoted as $e_i^p(x)$).

Only results from MOCell over the `ms2` instance will be analysed due to space constraints. All solutions of the approximated Pareto front obtained in different independent runs of MOCell are being considered. The $e_i^p(x)$ and $t_j^e(x)$ values have been computed for each employee and each task in all the solutions and the Spearman rank correlation coefficients [13] between all the $e_i^p(x)$, $t_j^e(x)$, $makespan(x)$ and $cost(x)$ have been calculated. The correlation coefficients are shown in Fig. 3. An arrow pointing up means positive correlation whereas an arrow pointing down means negative correlation. The absolute value of the correlation is shown in grey scale (the darker the higher).

Regarding the current values of $e_i^p(x)$ and $t_j^e(x)$ in all the solutions of all the independent runs of MOCell, $e_i^p(x)$ ranges between 1.00 and 1.61 with average values around 1.04. On the other hand, $t_j^e(x)$ ranges between 1 and 6 with average values around 1.56. This means that it is not common to have large working teams or a large number of parallel tasks per employee, therefore the

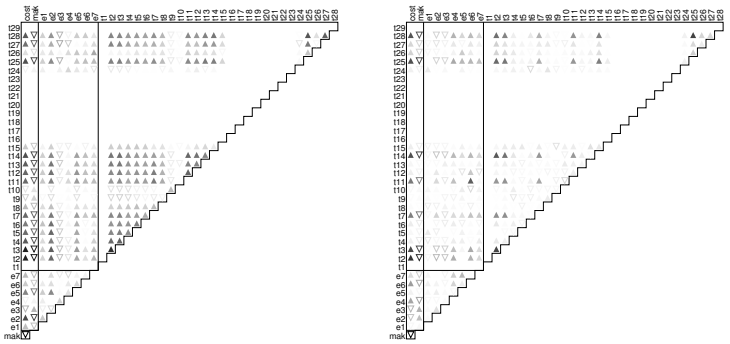

Fig. 3. Correlations between cost, duration, the number of average parallel tasks performed by the employees and the number of employees per task for the NR approach (left) and the STR approach (right) using MOCell. Solutions for `ms2` in the approximated Pareto front of all the independent runs.

communication overhead or the reduction of productivity due to parallel tasks is not high.

Focusing on the correlation between the make-span and the number of parallel tasks performed by the employees, a negative correlation with the exception of e_3 (and e_2 using the STR approach) can be observed. A negative correlation means that in order to reduce the make-span of the project, the employees will have to work in several tasks simultaneously. This seems to agree with common sense. Then, why does a positive correlation between make-span and employee e_3 appear? This employee is the only one able to do some tasks in the critical path of the project. Therefore, such critical tasks are assigned to this employee by the algorithm in order to reduce the execution time of the tasks. The above also explains the negative correlation between the size of the working teams $t_j^e(x)$ and $e_3^p(x)$. It is expected that in order to reduce the make-span the size of working teams must be increased, which also implies an increase in the number of parallel tasks each employee has to develop. This explains the positive correlation between $e_i^p(x)$ and $t_j^e(x)$ for the remaining employees.

Considering now the correlations between the make-span and the number of employees in each task, it is noticed, with no surprise, that reducing the make-span implies that more employees have to work on the tasks. However, some blank cells can be observed for which no correlation is detected. This happens in the tasks of the project for which only one employee has the required skills (non-zero productivity), like task t_1. This is just an illustration on how the analysis of solutions can provide some interesting information for the project manager.

4 Conclusions and Future Work

A new formulation of the Software Project Scheduling problem taking into account the productivity of the employees in developing different tasks of a software project and considering the inaccuracies of task cost estimations was presented. Experimental studies were carried out in order to analyse the performance of

four multi-objective algorithms on real-like instances for this problem. Solutions were analysed to illustrate the way project managers can use this tool to improve their decision making. Results show that MOCell is the best algorithm in solving this formulation of the problem, improving even the original solutions proposed by a project manager to the instances used in the experimental section. The analysis of the solutions reveals that the algorithms have been able to identify the tasks in the critical path and the most important employees for the project.

This work can be extended in several ways. An empirical study using real projects and their corresponding scheduling can be done with the help of data provided by software companies. Second, different robustness approaches can be used to take into account the inaccuracies in the productivity values. Third, new operators or search methods can be developed to improve the solutions or the required computational effort.

References

1. Chang, C.K., yi Jiang, H., Di, Y., Zhu, D., Ge, Y.: Time-line based model for software project scheduling with genetic algorithms. Information and Software Technology 50(11), 1142 – 1154 (2008)
2. Chicano, F., Luna, F., Nebro, A.J., Alba, E.: Using multi-objective metaheuristics to solve the software project scheduling problem. In: Proceedings of GECCO. pp. 1915–1922 (2011)
3. Coello Coello, C.A., Lamont, G.B., Van Veldhuizen, D.A.: Evolutionary Algorithms for Solving Multi-Objective Problems. Springer, New York, 2nd edn. (2007)
4. Deb, K.: Multi-objective optimization using evolutionary algorithms. John Wiley & Sons (2001)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Ev. Comp. 6(2), 182–197 (2002)
6. Garey, M.R., Johson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
7. Gutjahr, W., Katzensteiner, S., Reiter, P., Stummer, C., Denk, M.: Competence-driven project portfolio selection, scheduling and staff assignment. Central European Journal of Operations Research 16(3), 281–306 (September 2008)
8. Knowles, J.: A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. In: ISDA. pp. 552 – 557 (2005)
9. Knowles, J., Corne, D.: Approximating the nondominated front using the pareto archived evolution strategy. Evolutionary Computation 8(2), 149 – 172 (2000)
10. Moløkken, K., Jørgensen, M.: A review of surveys on software effort estimation. In: 2003 Int- Symp. on Empirical Software Engineering. pp. 223 – 231 (2003)
11. Nebro, A.J., Durillo, J.J., Luna, F., Dorronsoro, B., Alba, E.: A cellular genetic algorithm for multiobjective optimization. In: NICSO 2006. pp. 25–36 (2006)
12. Palpant, M., Artigues, C., Michelon, P.: LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. Annals of Operations Research 131, 237–257 (2004)
13. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. Chapman & Hall/CRC; 4 edition (2007)
14. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithms. In: EUROGEN 2001. pp. 95–100 (2002)
15. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE TEC 3(4), 257–271 (1999)