Achieving On-Time Delivery: A Two-Stage Probabilistic Scheduling Strategy for Software Projects

Xiao Liu[1], Yun Yang[1], Jinjun Chen[1], Qing Wang[2], and Mingshu Li[2]

[1] Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Melbourne, Australia 3122
{xliu,yyang,jchen}@swin.edu.au
[2] Laboratory for Internet Software Technologies, Institute of Software
Chinese Academy of Sciences
Beijing, 100080 P.R. China
{wq,mingshu}@itechs.iscas.ac.cn

Abstract. Due to the uncertainty of software processes, statistic based schedule estimation and stochastic project scheduling both play significant roles in software project management. However, most current work investigates them independently without an integrated process to achieve on-time delivery for software development organisations. For such an issue, this paper proposes a two-stage probabilistic scheduling strategy which aims to decrease schedule overruns. Specifically, a probability based temporal consistency model is employed at the first pre-scheduling stage to support a negotiation between customers and project managers for setting balanced deadlines of individual software processes. At the second scheduling stage, an innovative genetic algorithm based scheduling strategy is proposed to minimise the overall completion time of multiple software processes with individual deadlines. The effectiveness of our strategy in achieving on-time delivery is verified with large scale simulation experiments.

Keywords: Software Process, Schedule Estimation, Project Scheduling, Probabilistic Strategy, Genetic Algorithm.

1 Introduction

A software project is typically characterised by dynamic changes of the development environment and variant decisions of human stakeholders [13, 15]. Therefore, the estimation of software development schedule (and cost) with uncertainty as well as project scheduling under uncertainty are widely investigated and applied in software projects [6, 8]. But still, they are considered to be challenging issues for software development organisations of all sizes.

It has been witnessed that for a majority of software projects, on-time delivery of core capabilities is increasingly become the main focus of software processes in the dynamic business world nowadays [13]. Meanwhile, for many software development organisations, especially of small and medium sizes, their main business targets are short-term contracts from a relatively fixed group of customers in the market. These customers usually require the development of small scale software components within hard deadlines so as to meet their dynamic and urgent business needs. The competitive strength of these software development organisations critically relies on the ability of on-time delivery. Therefore, instead of pursuing multiple objectives such as reducing project schedule, budget and staff overload at the same time, this paper focuses on achieving on-time delivery. For software development organisations, an estimated software schedule serves as the guideline for project bidding and planning given specific customer needs and software process performance [14]. Furthermore, project scheduling is to decide "who does what" and optimise specific objectives, e.g. minimum schedule and budget. In recent years, great efforts have been dedicated to statistic based schedule estimation and stochastic project scheduling in software project management [6, 9]. However, most current work investigates schedule estimation and project scheduling independently without an integrated process. On one hand, estimated schedules for specific software processes cannot be realised without project scheduling to assign proper employees with proper tasks. On the other hand, without schedule estimation, project scheduling will not be effectively guided and probably result in frequent schedule overruns.

To achieve on-time delivery, this paper proposes a two-stage probabilistic project scheduling strategy to address the above issues. Specifically, at the first pre-scheduling stage for individual software processes, a probability based temporal consistency model is presented to facilitate a win-win negotiation between customers and project managers. This negotiation, namely pre-scheduling, supports the setting of balanced deadlines based on the process performance baseline. At the second scheduling stage for multiple software processes, an innovative genetic algorithm (GA) [1] based scheduling strategy which utilises a two-phase searching algorithm and a package based initialisation approach is proposed. The objective for the scheduling stage is to minimise the overall completion time of multiple software processes given that all individual software processes can be completed ahead of the deadlines as set at the pre-scheduling stage.

The remainder of the paper is organised as follows. Section 2 presents the related work and problem analysis. Section 3 gives the overview of our two-stage probabilistic project scheduling strategy. Section 4 presents the pre-scheduling stage and Section 5 proposes the scheduling stage. Section 6 describes large scale simulation experiments to verify the effectiveness of our strategy in achieving on-time delivery. Finally, Section 7 addresses our conclusions and future work.

2 Related Work and Problem Analysis

2.1 Related Work

With the dynamic nature of software development environments, various uncertainty and inconsistencies arise and thus bring challenges for schedule (and cost) estimation in software processes [15]. Jørgensen and Shepperd present a systematic review of software development schedule/cost estimation studies in [9]. For the 9 categories of research topics, estimation methods account for 61% of the samples and rank the first place. In recent years, various strategies such as feature prioritisation, core capability determination, risk driven strategies, earned value management, statistical process control and so forth, are presented to deal with uncertainty assessment and uncertainty control for software schedule (and cost) [13]. For schedule estimation of a specific software process, one of the most important concepts is Process Performance Baseline (PPB) [6]. PPB utilises two indicators, i.e. process performance and process capability. Here, process performance is “a measure of actual results achieved by following a process” and process capability is “the range of expected results that can be achieved by following a process”. In [14], a statistic based approach is proposed to establish and refine software process performance baseline where the average value μ and the standard deviation σ of data samples are defined as process performance and capability respectively. Specifically, process performance baseline is normally controlled under the limits of $\mu \pm 3\sigma$.

Project scheduling is to allocate proper tasks to proper employees or subcontractors in order to result in successful projects. Due to the uncertainty of software projects, most project scheduling strategies aim to generate a feasible schedule within given constraints, such as schedule and budget, to serve as a baseline schedule for real project execution. Five fundamental approaches for scheduling under uncertainty are identified in [6], i.e. reactive scheduling, stochastic project scheduling, fuzzy project scheduling, proactive scheduling and sensitivity analysis. Specifically, stochastic project scheduling strategies mainly concern about scheduling project tasks with uncertain durations in order to minimise the expected project schedule [3, 16]. Among them, genetic algorithm, as a commonly applied heuristic method, is often employed to solve complicated optimisation problems in resource constrained scheduling problems [12]. An empirical study in [7] demonstrates that the evolution algorithm is capable of finding the best-known solutions in 68% of the 2370 instances with an average overall error rate of 0.95%. [3] proposes a time-line based model as well as 8 heuristic rules to simulate real-world situations to enhance the ability of GA. In [1], structured studies have shown that GA is flexible and accurate for many different software project scenarios.

2.2 Problem Analysis

For a specific software project, schedule estimation and project scheduling are two fundamental issues for achieving on-time delivery. In practice, for project bidding and negotiation, project managers usually need to estimate project schedules based the statistic performance of software development organisations. During this period of time, project deadlines are set based on estimated schedules. However, practical data show that about one-third of the projects exceed their estimated schedules by 25% [13]. Two of the critical problems cause schedule overruns are as follows.

1) A project schedule is not well balanced between the software process performance baseline and customer needs. If project deadlines are far beyond the software process performance baseline, schedule overruns are highly expected. Therefore, in practice, some robust project scheduling strategies such as temporal protection intentionally extends the statistic task durations to protect the baseline schedule from resource breakdowns [6]. Meanwhile, most work only emphasises the role of project managers to estimate schedules and set project deadlines for individual software processes. However, the deadlines may often be set unrealistically tight due to customers’

pressure and thus results in frequent schedule overruns. Therefore, compromised project schedules which achieve proper balance between the software process performance baseline and customer needs are certainly more desirable.

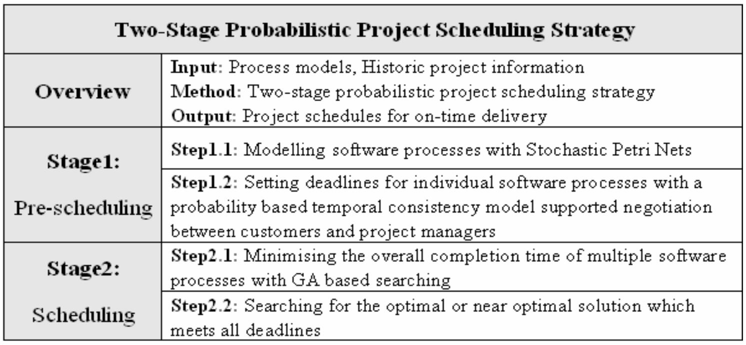
2) An individual baseline schedule does not consider the situation of multiple software processes [3]. One of its priorities for project scheduling is to ensure that software processes can be completed within specific deadlines. However, since software process performance is heavily dependent on the people who execute the process rather than the device of the product line, one of the major reasons deteriorates the performance and causes significant delays is the competition of employees among multiple software processes. In fact, this problem frequently occurs in a software development organisation and causes serious overruns even though project schedules are estimated based on the performance baseline. In another word, without considering the situation of multiple software processes, baseline schedules for individual software processes cannot guarantee the success of on-time delivery.

To tackle the above two problems, joint efforts need to be dedicated by schedule estimation and project scheduling as an integrated project scheduling strategy.

3 A Two-Stage Probabilistic Project Scheduling Strategy

In this section, we present the overview of our two-stage probabilistic project scheduling strategy. As depicted in Table 1, our project scheduling strategy consists of two stages, i.e. the pre-scheduling stage and the scheduling stage.

Table 1. Project Scheduling Strategy



Two-Stage Probabilistic Project Scheduling Strategy	
Overview	**Input**: Process models, Historic project information **Method**: Two-stage probabilistic project scheduling strategy **Output**: Project schedules for on-time delivery
Stage1: Pre-scheduling	**Step1.1**: Modelling software processes with Stochastic Petri Nets
	Step1.2: Setting deadlines for individual software processes with a probability based temporal consistency model supported negotiation between customers and project managers
Stage2: Scheduling	**Step2.1**: Minimising the overall completion time of multiple software processes with GA based searching
	Step2.2: Searching for the optimal or near optimal solution which meets all deadlines

At the pre-scheduling stage, the main objective is to set balanced deadlines for individual software processes. At this stage, Step 1.1 is to model software processes with Stochastic Petri Nets [2] which will be introduced later in this section as the specification tool. Step 1.2 is to set balanced deadlines for individual software processes with a probability based temporal consistency model which is to support a win-win negotiation between customers and project managers. At the scheduling stage, the main objective is to generate a scheduling plan which assigns proper tasks to proper employees in order to achieve on-time delivery. At this stage, an innovative GA based scheduling strategy is conducted as Step 2.1 to minimise the overall completion time of multiple software processes. Step 2.2 is to search for the optimal or near optimal solution (i.e. the best or near best project scheduling plan) which meets the deadlines for individual software processes set at the pre-scheduling stage.

Technical details of pre-scheduling and scheduling will be presented in Section 4 and Section 5 respectively. Here, we introduce Stochastic Petri Nets (SPN) to specify software processes. SPN can provide powerful abstractions of such as control flows, underlying resources, stochastic temporal information. Besides conventional Petri Nets notations of place, transition and arc, an additional new notation of task weight is employed for specifying statistic task durations. As proposed in [11], task weight is defined with the probability and statistic iteration times of each task based on four fundamental control flow patterns, i.e. sequence, choice, parallelism and iteration. Specifically, for a sequence process, the task weight is specified as 1. For a choice process, the task weight is equal to the probability of the path to be chosen. For a parallelism process, the task weight of the path with largest expected duration is defined as 1 while others are defined as 0 since they do not dominate the overall execution time. For an iteration process, the task weight is defined as the statistic iteration times. The duration distribution is associated with each task based on the statistic performance. The purpose of modelling software processes with SPN is to reflect the stochastic temporal information to support the schedule estimation and project scheduling.

4 Pre-scheduling for Individual Software Processes

The temporal consistency model is defined to quantitatively measure the state of specific processes given specified temporal constraints [4, 5]. In this paper, we employ a probability based temporal consistency model to support deadline setting. The probability based temporal consistency model is defined based on the concept of weighted joint normal distribution [11]. Weighted joint normal distribution can be used to analyse the distribution of the overall completion time, namely the performance baseline for a specific process based on the distribution of individual task durations. Here, we first define some notations. For a specific task t_i, its maximum duration and minimum duration are defined as $D(t_i)$ and $d(t_i)$ respectively. In addition, we employ the "3σ" rule which has been widely used in statistical analysis [10]. The "3σ" rule depicts that for any sample coming from normal distribution model, it has a probability of 99.73% to fall into the range of $[\mu - 3\sigma, \mu + 3\sigma]$ where μ is the average value and σ is the standard deviation. In this paper, we define the maximum duration as $D(t_i) = \mu_i + 3\sigma_i$ and the minimum duration as $d(t_i) = \mu_i - 3\sigma_i$. Meanwhile, for a software process SP which consists of n tasks, its deadline is denoted as $F(SP)$ and its start time is denoted as $S(SP)$. A deadline of $F(SP)$ is a fixed time by which the process SP must be completed [5]. Now, based on [11], we present the definition of probability based temporal consistency model on fixed time temporal constraints (deadlines) as follows.

Definition: (Probability based temporal consistency model)

For a software process SP with a deadline of $F(SP)$ that starts at $S(SP)$, $F(SP)$ is said to be of:

1) Absolute Consistency (AC), if $\sum_{i=1}^{n} w_i(\mu_i + 3\sigma_i) < F(SP) - S(SP)$ *;*

2) Absolute Inconsistency (AI), if $\sum_{i=1}^{n} w_i(\mu_i - 3\sigma_i) > F(SP) - S(SP)$ *;*

3) $\alpha\%$ *Consistency (* $\alpha\%$ *C), if* $\sum_{i=1}^{n} w_i(\mu_i + \lambda\sigma_i) = F(SP) - S(SP)$ *.*

Here, w_i *stands for the weight of task* t_i, λ *(* $-3 \le \lambda \le 3$ *) is defined as the* $\alpha\%$ *confidence percentile with the cumulative normal distribution function of*

$$f(\mu_i + \lambda\sigma_i) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\mu_i+\lambda\sigma_i} e^{-(x-\mu_i)^2 / 2\sigma_i^2} \bullet dx = \alpha\% \; (0 < \alpha < 100).$$

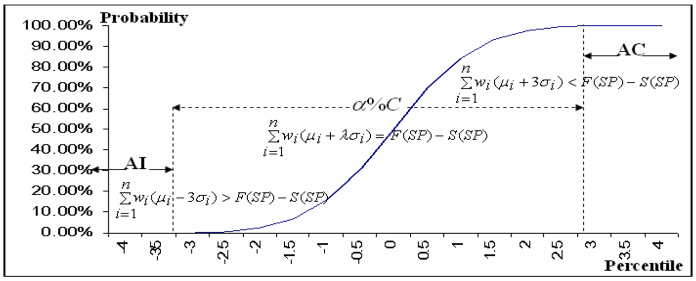


Fig. 1. Probability Based Temporal Consistency

With the above model, different states of the software process towards a specific deadline are described with continuous probability values. As shown in Figure 1, these values form a continuous Gaussian curve. According to the probability based temporal consistency model, the basic performance baseline (average activity duration) can only guarantee a probability consistency state of 50% which is normally much lower than the customer's minimal confidence. Therefore, a more realistic deadline acceptable to all stakeholders needs to be negotiated [11].

Now we demonstrate the win-win negotiation for setting balanced deadlines. Here, we denote the obtained weighted joint distribution of the target software process SP as $N(\mu_{sp}, \sigma_{sp}^2)$ and assume the minimum threshold to be $\beta\%$ for the probability consistency which implies the customer's acceptable bottom-line probability for on-time delivery. The actual negotiation starts with the customer's initial suggestion of a deadline of $F(SP)$ and the evaluation of the corresponding temporal consistency state by the project manager. If $F(SP) - S(SP) = \mu_{sp} + \lambda\sigma_{sp}$ with λ as the $\alpha\%$ percentile, and $\alpha\%$ is below the threshold of $\beta\%$, then the deadline needs to be adjusted, otherwise the negotiation terminates. For the negotiation, the subsequent process is the iteration that the customer proposes a new deadline which is less restricted as the previous one and the project manager re-evaluates the consistency states, until it reaches or is above the minimum probability threshold [11]. This win-win negotiation, i.e. pre-scheduling process, facilitates the setting of balanced project deadlines.

5 Scheduling for Multiple Software Processes

To achieve on-time delivery, our scheduling objective is to minimise the overall completion time of multiple software processes given their individual deadlines set at the pre-scheduling stage can be met. For such an objective, we propose an innovative GA based project scheduling strategy. GA is a class of evolutionary algorithms inspired by evolutionary biology [1, 16]. In GA, three basic GA operations, i.e. selection, crossover and mutation, are conducted to imitate the evolution process in nature. After the stopping condition is met, the solution with the best fitness value which represents the optimal or near-optimal solution is returned [3].

Here, in order to enhance the performance and satisfy real-world situations, we first identify two critical aspects which should be tackled in GA based project scheduling.

(1) How to achieve on-time delivery for individual software processes while minimising the overall completion time for multiple software processes.

As discussed in Sections 1 and 2, it is a priority to ensure on-time delivery. However, for software development organisations, it is also important to minimise the overall completion time for multiple software processes in order to reduce the project cost. Therefore, how to effectively balance these two objectives is a critical aspect.

(2) How to implement heuristic rules in GA for practical restrictions.

In real-world software projects, there are many restrictions which affect the tasks-to-employees assignment. Therefore, in order to support decision making under more realistic conditions, many heuristic rules such as resource continuity (e.g. assigning a group of highly related tasks to a fixed employee so as to reduce the overhead of task transfer), adjustment of workload and overstaffed projects, are supplemented [3]. However, how to implement these heuristic rules in GA is a challenging issue.

```
Strategy:GA based project scheduling
Input: Software process models
       Employee models
Output: Generated Scheduling Plans
//two dimensional encoding
1) ENCODING(L);
// package based generation of initial populaton
2) INITIALISATION(Population)
// Optimising overall completion time
3) While (maximum generation is not reached)
   {
    //selecting Parents from Population
4)    SELECTION(Population) ;
5)    CRSSSOVER(Parents)→Children;
    //change resource for a random task
6)    MUTATION(Children);
    //check with software process models and heuristic rules
7)    VALIDATION(Children);
    //store the best child into a solution set; replace worst child
    //with the best child
8)    STORE(BestChild, SolutionSet);
9)    REPLACE(WorstChild, BestChild);
   }
10) Return SolutionSet and BestChild in Population.
// Searching for the BestSolution from solution set
11) BestSolution =BestChild;
12) While (not end of the SolutionSet)
   {
    //Compare each solution and set BestSolution as the one meets all
    //individual deadlines and has smaller overall completion time
13)    BestSolution=COMPARE(BestSolution, Child);
   }
14) Return the BestSolution;
// decoding and update
15) DECODING(BestSolution);
16) UPDATE(L).
```

(a) GA Based Project Scheduling Strategy

```
Function: INITIALISATION
Input: Software process models $SP\{SP_i(1 \le i \le k)\}$
       Employee models $EP\{EP_i(1 \le i \le p)\}$
Output: $sched_i, alloc_i$ : chromosome
1) for i= 1 to size   // generate a fixed size of solutions
// generate a single solution
2)     for each Software Process SP
// arrange tasks to form packages
// function size() returns the number of tasks in each SP
3)         $sched_i\left(\sum_{m=1}^{j-1} size(SP_m)+1 : \sum_{m=1}^{j} size(SP_m)\right)$
               $= task\left(\sum_{m=1}^{j-1} size(SP_m)+1\right) : task\left(\sum_{m=1}^{j} size(SP_m)\right);$
// allocate employees to each package
4)         for each Package P in each Software Process SP
// check package requirements with employee models and
// return the matched employee set $EP'\{\}$
5)             $EP'\{\} = Check(Package, EP\{\})$
// apply heuristic rules $R\{\}$ and return the valid employee set $EP''\{\}$
6)             $EP''\{\} = Check(R\{\}, EP'\{\})$
// random allocate valid employees to the package
7)             $alloc_i\left(\sum_{m=1}^{j-1} size(SP_m)+1 : \sum_{m=1}^{j} size(SP_m)\right)$
               $= random(EP''\{\});$
8)         end
9)     end
10) end
```

(b) Package Based Initialisation

Fig. 2. Pseudo-code for GA based Project Scheduling Strategy

In our GA based scheduling strategy, we propose a two-phase searching algorithm to address the first aspect and a package based initialisation approach to address the second aspect. The pseudo-code for our GA based project scheduling strategy is presented in Figure 2 where Figure 2(a) presents the scheduling strategy and Figure 2(b) presents the package based initialisation approach. As shown in Figure 2(a), our scheduling strategy has two main input parameters, i.e. software process models and the employee models. Here, software processes models are specified with SPN as introduced in Section 3. Each software process model describes the control flows, namely the precedence relationships between tasks. Meanwhile, based on the results of the pre-scheduling stage, stochastic temporal information such as process deadlines and the statistics of task durations are also provided. The employee models mainly define the specific skills possessed by individual employees and some other information related to the decisions on task assignment such as his/her proficiency level (measured in execution speed), payment rate, workload, project experience and so on. Our strategy starts with the encoding of an empty task-employee list and generation of initial population with package based initialisation (line 1 and line 2). Line 3 to line 14 is the GA based two-phase searching algorithm to find the best solution. Finally, the best solution is decoded (line 15) and the task-employee list is updated (line 16). After this scheduling process, the task-employee list which represents the optimal or near-optimal scheduling plan is generated.

Two-phase searching algorithm. To address the first aspect identified, our GA based scheduling strategy adopts a two-phase searching algorithm as depicted in Figure 2(a). The first phase (line 3 to line 10) is to optimise the overall completion time of multiple software processes. Based on genetic operations, e.g. selection, crossover and mutation (line 4 to line 6), the searching space is expanded and solutions with higher fitness values are found. Here, the fitness value is defined according to the overall completion time of all the software processes. The smaller the overall completion time, the higher the fitness value is. The function of validation (line 7) is to check the generated solutions if they are correct with restrictions, e.g. the precedence relationships and other heuristic rules. During each generation, the best child is stored in a solution set (line 8) and the worst child is replaced by the best one (line 9). The second phase (line 11 to line 14) is to search for the best solution from the whole solution set composed of the best child in each generation produced in the first searching phase. The best solution found should meet the deadlines of individual software processes while having the minimum overall completion time. Our two-phase searching algorithm guarantees the return of balanced solutions. The reason is that on one hand, a vast number of possible solutions are generated and evaluated in the first searching phase. On the other hand, the best child with the minimum overall completion time of each generation is recorded in the solution set. Therefore, if a balanced solution cannot be found, especially after huge numbers of generations, we are able to claim that it is not possible to find a valid solution which can achieve on-time delivery for all software processes. Otherwise, a balanced scheduling plan which meets the deadlines of individual software processes should be found in the solution set. However, to support decision making, highly ranked solutions in the solution set will be returned instead in this case where no best solution exists. The project managers can make further decisions, e.g. recruitment of more employees or outsource to subcontractors, to ensure the success of on-time delivery.

Package based initialisation approach. As the second aspect identified, practical restrictions in the real-world software projects should be considered in the scheduling strategy so as to support and satisfy realistic conditions. For such a purpose, we propose an innovative package based initialisation approach based on two dimensional encoding. As depicted in Figure 3, the first dimension $Sched_i$ denotes scheduled tasks and the second dimension $alloc_i$ denotes the allocated resources (employees or subcontractors). As discussed in many literatures, GA initialisation for the initial population is critical towards the outcomes of GA [16]. Basically, the initial population should be valid and effective. In our scenario, to be valid, the initial population should assign specific tasks to valid employees who possess the ability to fulfil these tasks. Meanwhile, for a specific employee, the tasks assigned to him/her should be conformed to their precedence relationships defined in software process models. To be effective, more restrictions, e.g. resource continuity and the 8 heuristic rules proposed in [3], should be applied for practical project management. For such an issue, a package based random initialisation approach is to support the generation of population which is both valid and effective. 'Package' here denotes a group of highly related tasks in the same software process, which can be defined by experienced project managers, with correct precedence relationships. Meanwhile, these tasks share the same employees allocated to each package with the enforcement of restrictions. The design of a package here not only ensures the correct task flows for a software process since tasks with correct precedence relationships are assigned to the same employees, but also is capable of facilitating resource continuity and other heuristics. For a specific package, a set of employees are formed first by checking required abilities (line 5 of Figure 2(b)). Afterwards, a further checking process is applied to select valid employees based on enforced heuristics (line 6 of Figure 2(b)). Finally, one of the valid employees is randomly assigned to this package (line 7 of Figure 2(b)). As shown in Figure 3, for the employee assignment of n tasks, an employee set, say $\{R_1, R_2, R_3, \ldots R_p\}$ is first formed. After that, specific heuristic rules are applied. For example, one of the employees, say R_2, which currently has the lowest workload, is assigned to Package 1. Following a similar way, a valid and effective solution comprised of Packages 1, 2, ..., k (Figure 3) is automatically generated (loop of line 2 of Figure 2(b)). The initial population is formed by a fixed size of automatically generated solutions (loop of line 1 of Figure 2(b)).

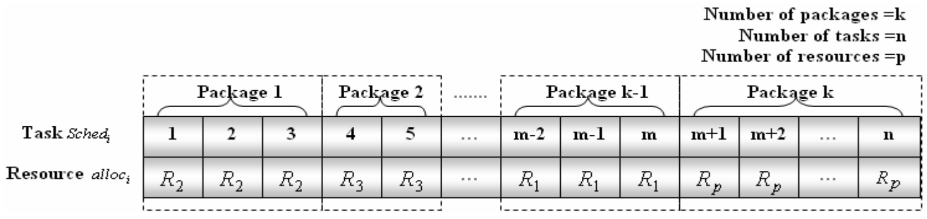
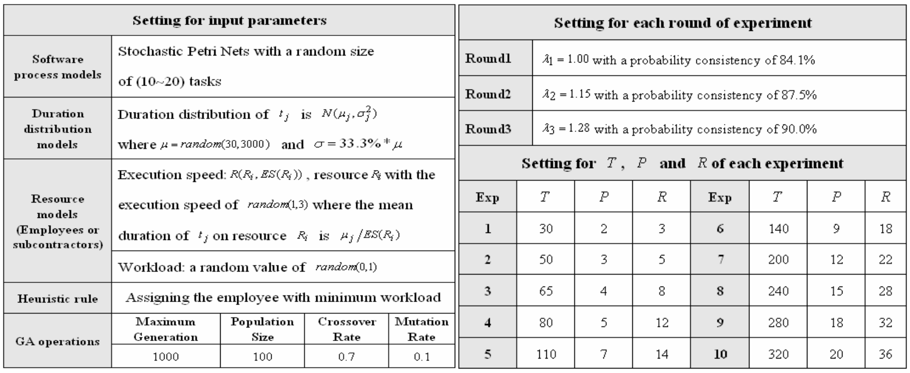


Fig. 3. Package Based Initialisation Approach

6 Evaluation

In this section, we evaluate the effectiveness of our two-stage probabilistic project scheduling strategy through large scale simulation experiments where the simulation results are independent of specific platforms. The settings for simulation experiments are presented in Table 2. We manually generate stochastic Petri Nets with a random size of 10 to 20 tasks. To simplify our simulation scenario so as to focus on the objective of on-time delivery, we only set two attributes for employee models, i.e. the execution speed (measured for proficiency level) and the workload. The execution speed of each employee is randomly specified from a range of 1 to 3 where 1 denotes that the mean execution time equals to the expected duration while 3 denotes that the mean execution time equals to the expected duration divided by 3. The workload of each employee is a random value from 0 to 1 where 0 means the employee has not been assigned with any tasks and 1 means the employee is fully occupied and cannot be assigned with more tasks. For each task assignment to a specific employee, his/her workload increases by 0.1 so an employee can take up to 10 tasks. Accordingly, we adopt one simple heuristic rule that is assigning the current task to the employee with minimum workload. As for the settings of GA operations, maximum generation is used as the stopping condition and its value is 1000. The initial population size is set as 100. The crossover rate and mutation rate are set as 0.7 and 0.1 respectively as common practice. To evaluate our strategy with large scale simulation experiments, we conducted 3 rounds, i.e. COM(1.00), COM(1.15) and COM(1.28) with different λ as 1.00, 1.15 and 1.28 to reflect different pre-scheduling results with 84.1%, 87.5% and 90.0% consistency respectively. As depicted in Table 3, within each round, we conducted 10 experiments with different settings of number of tasks (T) with a range of 30-320 in total whilst 10-20 for each process, the number of processes (P) with a range of 2-20, and the number of resources (R) with a range of 3-36.

Table 2. Settings for Simulation Experiments



Setting for input parameters				
Software process models	Stochastic Petri Nets with a random size of (10~20) tasks			
Duration distribution models	Duration distribution of t_j is $N(\mu_j, \sigma_j^2)$ where $\mu = random(30, 3000)$ and $\sigma = 33.3\% * \mu$			
Resource models (Employees or subcontractors)	Execution speed: $R(R_i, ES(R_i))$, resource R_i with the execution speed of $random(1,3)$ where the mean duration of t_j on resource R_i is $\mu_j / ES(R_i)$			
	Workload: a random value of $random(0,1)$			
Heuristic rule	Assigning the employee with minimum workload			
GA operations	Maximum Generation	Population Size	Crossover Rate	Mutation Rate
	1000	100	0.7	0.1

Setting for each round of experiment	
Round1	$\lambda_1 = 1.00$ with a probability consistency of 84.1%
Round2	$\lambda_2 = 1.15$ with a probability consistency of 87.5%
Round3	$\lambda_3 = 1.28$ with a probability consistency of 90.0%

Setting for T, P and R of each experiment							
Exp	T	P	R	Exp	T	P	R
1	30	2	3	6	140	9	18
2	50	3	5	7	200	12	22
3	65	4	8	8	240	15	28
4	80	5	12	9	280	18	32
5	110	7	14	10	320	20	36

Two attributes, i.e. the *improvedPercent* (the difference of the overall completion time before and after our GA based scheduling strategy divided by the one before) and the *overrunRate* (the number of processes which fails to be completed within deadlines divided by the number of total processes), are investigated. Here, for fairness, the overall completion time before our scheduling strategy is specified as the average completion time of all valid solutions generated in the GA based searching phase. Meanwhile, we also investigate the *overrunRate* of the initial project scheduling plans generated by package based initialisation but without the GA based searching phase, namely a NIL strategy, for the purpose of comparison. To investigate the statistic performance, each experiment is executed for 100 times. Therefore, in our simulation, 3 rounds, 10 experiments and 100 execution times, namely a large scale of 3000 independent experiments, have been executed.

The simulation results are presented in Figure 6. As can be seen from the left subplot in Figure 6, the mean *improvedPercent* and the number of tasks are roughly on the same trend. This verifies the effectiveness of our strategy in scheduling multiple software processes and optimising their overall completion time. Furthermore, our strategy performs even better when the scheduling scenario becomes more complicated as shown by increasing *improvedPercent*. We also note that despite the increase of probability consistency, i.e. from 84.1%, 87.5% to 90.0%, which means less restricted deadlines, the mean *improvedPercent* does not vary much. Hence, a larger probability consistency does not guarantee a better *improvedPercent.* As for the *overrunRate* depicted in the right subplot of Figure 6, the simulation results show that if without GA based searching phase, i.e. the NIL strategy, the average *overrunRate* is high with a growing trend based on the increase of the number of software processes. However, with our scheduling strategy, the *overrunRate* is much lower as depicted. In this case, the increase of probability consistency results in the lower mean *overrunRate.* To conclude, we can claim that our two-stage probabilistic scheduling strategy is effective for achieving on-time delivery.

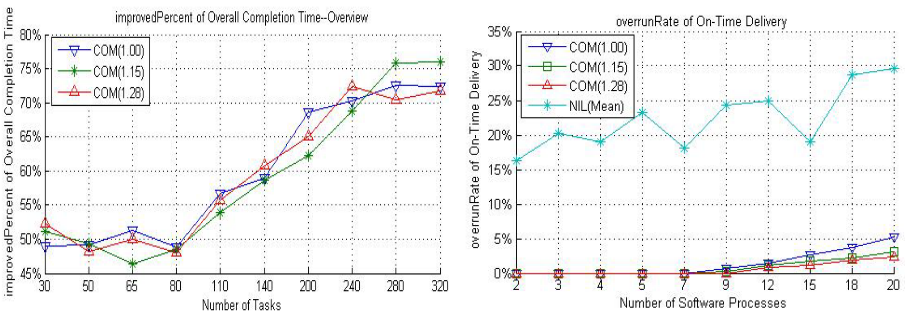


Fig. 6. Simulation Results

7 Conclusions and Future Work

In this paper, we have proposed a two-stage probabilistic scheduling strategy which integrates statistic based schedule estimation and stochastic project scheduling in order to achieve on-time delivery of software projects. Our strategy aims to generate the best scheduling plan which can meet the deadlines of individual software processes while having the minimum overall completion time of multiple software processes.

Hence, at the first pre-scheduling stage, a probability based temporal consistency model has been presented to facilitate a win-win negotiation between customers and project managers for setting balanced project deadlines based on individual software process performance baselines. Given these deadlines, at the second scheduling stage, an innovative GA based project scheduling strategy which utilises a two-phase searching algorithm and a package based initialisation approach has been proposed to search for the best scheduling plan under the resource constraint of the current software development organisations. Based on the results of large scale simulation experiments, it has been verified that the best scheduling plan can be found in most cases with our project scheduling strategy. However, even in the case where such solution does not exist due to the resource constraint, the generated solutions can still support the project manager to make further decisions such as recruitment of employees or outsourcing to ensure the success of on-time delivery.

In the future, to tackle the case where the best scheduling plan can not be found, we will try to identify the key software processes where on-time delivery can be achieved with minimum increase of extra cost.

References

1. Alba, E., Chicano, J.F.: Software Project Management with GAs. Information Sciences 177, 2380–2401 (2007)
2. Bucci, G., Sassoli, L., Vicario, E.: Correctness Verification and Performance Analysis of Real-Time Systems Using Stochastic Preemptive Time Petri Nets. IEEE Transaction on Software Engineering 31(11), 913–927 (2005)
3. Chang, C.K., Jiang, H., Di, Y.: Time-line Based Model for Software Project Scheduling with Genetic Algorithms. Information and Software Technology 50, 1142–1154 (2008)
4. Chen, J., Yang, Y.: Adaptive Selection of Necessary and Sufficient Checkpoints for Dynamic Verification of Temporal Constraints in Grid Workflow Systems. ACM Transaction on Autonomous and Adaptive Systems 2(2) Article 6 (2007)
5. Chen, J., Yang, Y.: Temporal Dependency based Checkpoint Selection for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In: Proceedings of 30th International Conference on Software Engineering, Leipzig, Germany, pp. 141–150 (2008)
6. Herroelen, W., Leus, R.: Project Scheduling Under Uncertainty: Survey and Research Potentials. European Journal of Operational Research 165, 289–306 (2005)
7. Hindi, K.S., Yang, H., Fleszar, K.: An Evolutionary Algorithm for Resource-Constrained Project Scheduling. IEEE Transaction on Evolutionary Computation 6(5), 512–518 (2002)
8. Jørgensen, M.: Evidence-Based Guidelines for Assessment of Software Development Cost Uncertainty. IEEE Transaction on Software Engineering 31(11), 942–954 (2005)
9. Jørgensen, M., Shepperd, M.: Systematic Review of Software Development Cost Estimation Studies. IEEE Transaction on Software Engineering 33(1), 33–53 (2007)
10. Law, A.M., Kelton, W.D.: Simulation Modelling and Analysis, 4th edn. McGraw-Hill, New York (2007)
11. Liu, X., Chen, J., Yang, Y.: A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 180–195. Springer, Heidelberg (2008)

12. Tracy, D.B., Howard, J.S., Noah, B.: Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing 61(6), 810–837 (2001)
13. Wang, Q., Jiang, N.: Practical Experience of Cost/Schedule Measure through Earner Value Management and Statistical Process Control. In: Proceedings of 2006 International Software Process Workshop, Shanghai, China (2006)
14. Wang, Q., Jiang, N., Gou, L., Liu, X., Li, M., Wang, Y.: BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline. In: Proceedings of 28th International Conference on Software Engineering, Shanghai, China, pp. 584–594 (2006)
15. Yang, Q., Li, M., Wang, Q.: An Algebraic Approach for Managing Inconsistencies in Software Processes. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 121–133. Springer, Heidelberg (2007)
16. Zhang, D., Tsai, J.P.: Machine Learning Applications in Software Engineering. World Scientific, Singapore (2005)