# Multi-objective Dynamic Software Project Scheduling: An Evolutionary Approach for Uncertain Environments

by

Natasha Nigar


A thesis submitted to the
University of Birmingham for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham

December 2020

# Abstract

Software project scheduling (SPS) in an uncertain and dynamic environment is critical for within budget and on time completion of real-world projects. This is an important issue in the software engineering practices, as budget and effort must be managed well enough for successful project completion. This deals with suitable allocation of employees to tasks in a software project with an aim to minimize project cost and schedule. Most of the existing research address this problem by considering only static and deterministic scenarios with no dynamic disruptions. However, the SPS has emerged as a dynamic scheduling problem which is challenged by inherent agility for medium to large scale projects. It requires not only the capability to handle dynamic events but also multiple objectives to ensure successful project completion. The increasing trend of cloud based software as a service (SAAS) solutions (large-scale complex projects) also requires projects on schedule and within budget.

In this thesis, we address this challenge (handling of dynamic events potentially capable to disrupt project quality, schedule, and cost) by formulating the software project scheduling problem as an optimization problem that regenerates a feasible software project schedule. We present three multi-objective (project duration, cost, robustness, and stability) models, subjected to variety of practical constraints, to deal with dynamic events. First, we present a model to deal with 'Employee Turnover' dynamic event that handles both employee rescheduling and recruitment scenarios, along with a multi-objective evolutionary algorithm. Second, we tackle another dynamic event 'new employee addition' by proposing a multi-objective evolutionary algorithm as novel heuristic approach. Third, we propose a new model for the SPS problem by considering an important human factor i.e. 'employee experience'.

The work in this thesis presents our first attempts to address some of the most challenging problems in software project scheduling under dynamic environment. The proposed evolutionary algorithmic approaches and models have been evaluated against existing state-of-the-art algorithms.



***Keywords***: dynamic software project scheduling, multi-objective optimization, mathematical model, metaheuristics.

*To my family.*

# Acknowledgement

First and foremost, I would like to express my deepest and sincere gratitude to my supervisors Prof. Xin Yao and Dr. Miqing Li who advised, helped and supported me during my PhD studies. They are best mentors one could ever expect for. Prof. Xin Yao is very responsible and strives for excellence. Despite his very busy schedule, he always made time to discuss my research and provided useful comments and suggestions that improved our work. Miqing is very kind and always guided me to whatever problems I met. He was always available to help, even on holidays and supported me in every critical situation. I would not have widen the scope of my research and gained more knowledge and skills without them. Beside my supervisors, I would like to thank all the respected academics I worked with during my research for their constructive feedback. I would like to thank Prof. Ata Kaban and Dr. Rami Bhasoon - my thesis group members - for their continuous feedback regarding my research. I would also like to thank Dr. Leandro Minku for his time and fruitful discussion regarding my research.

Sincerely, I would like to acknowledge the University of Engineering and Technology, Pakistan for their scholarship grant that funded my studies.

Last but not the least comes my family. Without my parents' belief in me, tolerance for my worries, and support in every possible way, I would not have made it to writing those lines. My kids and husband, who had to go through hard times to fulfil my dream for doctoral degree. My uncle who always supported and encouraged me in difficult times. For this and much more, I could not be more grateful for my family.

# Contents

# List of Figures

# List of Tables

# Nomenclature

## Acronyms

| | |
|---|---|
| CPM | critical path method |
| DE | dynamic event |
| DM | decision maker |
| DSPS | dynamic software project scheduling |
| EA | evolutionary algorithm |
| EMO | evolutionary multi-objective optimisation |
| HV | hypervolume |
| HVR | hypervolume ratio |
| MOEA | multi-objective evolutionary algorithm |
| MOP | multi-objective optimisation problem |
| PERT | program evaluation review technique |
| PM | person-month |
| PSO | particle swarm optimization |
| RCPSP | resource-constrained project scheduling problem |
| SBSE | search based software engineering |
| SBX | simulated binary crossover |
| SDLC | software development life cycle |
| SOA | state-of-the-art |
| SPM | software project management |
| SPS | software project scheduling |
| SPSP | software project scheduling problem |
| SSPS | static software project scheduling |
| TPG | task precedence graph |

## Algorithms

| | |
|---|---|
| BCE | bi-criterion evolution algorithm |
| EA | evolutionary algorithm |
| $\epsilon$-MOEA | $\epsilon$-dominance based MOEA |
| h_NEA | heuristic for new employee addition |
| NSGA-II | non-dominated sorting genetic algorithm II |
| NSGA-III | non-dominated sorting genetic algorithm III |
| OMOPSO | improved PSO using crowding, mutation and $\epsilon$-dominance concepts |
| SMPSO | speed-constrained multi-objective PSO |
| Two_Arch2 | improved two-archive algorithm |

## Symbols

| | |
|---|---|
| $A$ | archive set |
| $e_i$ | the $i$th employee |
| $E$ | set of employees |
| $f_i$ | the $i$th objective value |
| $m$ | number of objectives |
| $N$ | size of population |
| $n$ | decision variables number |
| $P$ | size of evolutionary population |
| $p_c$ | crossover probability |
| $p_m$ | mutation probability |
| $SK$ | employee skills set |
| $T$ | set of tasks |
| $t_j$ | the $j$th task |
| $t_0$ | the starting time |
| $t'$ | the scheduling point time |
| $x_{ij}$ | the degree of dedication of $i$th employee to $j$th task |
| $X$ | dedication matrix |
| $\varepsilon$ | employee's experience |
| $\prec$ | to Pareto dominate |

# Chapter 1

# Introduction

This thesis addresses the software project scheduling problem by applying Search Based Software Engineering (SBSE) approaches in a dynamic environment. This chapter presents the motivation that led to undertaking this challenge followed by research methodology, research questions and a brief summary of contributions. This chapter concludes with the overall structure of the thesis and research publications.

## 1.1 Search-based Software Engineering

Search-based software engineering (SBSE) has emerged as a new field of software engineering research and practice [78]. In general, SBSE is known as sub-field of software engineering that seeks to reformulate software engineering complex issues as search problems and applies metaheuristic techniques to solve them. Software engineers typically faced with multiple objectives that must be delivered under various constraints. Balancing all these is a critical task. SBSE has become an increasingly popular paradigm for solving such highly complex and dynamic software engineering problems which otherwise are infeasible to solve. This nature of problem often makes the definition of analytical algorithms problematic. However, the SBSE approaches tend to generate a pool of candidate solutions automatically and gradually evolve them to be increasingly more desirable. As a result, these algorithms find optimal, near-optimal or those solutions which fall within a specified acceptable tolerance of a perceived ideal. Such factors

make robust search-based optimization techniques, e.g., evolutionary algorithms (EAs) [86], simulated annealing [93], and tabu search, [69] readily applicable.

An evolutionary algorithm dealing with problems considering multiple objectives is known as multi-objective evolutionary algorithm (MOEA), and such problems are known as multi-objective problems (MOPs). In MOPs, objectives are typically of conflicting nature and, therefore, we obtain a set of optimal solutions rather a single solution. These solutions are known as Pareto optimal solutions [50]. It indicates that the search space has no other solutions that are superior for all the objectives considered. A simplified flow chart of EAs working is presented in Figure 1.1. The EA starts with the population initialization. The generated solutions are evaluated based on some fitness criteria. New generations are produced by applying selection, crossover and mutation operators. The algorithm stops when stop criterion (iterations) is met.

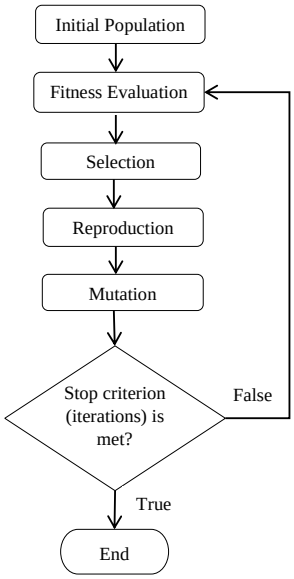



Figure 1.1: Basic Flow Chart of an Evolutionary Algorithm

By reviewing the literature, evolutionary algorithms have been identified to be used widely by researchers to tackle complex software engineering problems as shown in Figure 1.2. A brief explanation of these problems is given below. This research focuses on software project scheduling problem which has been described in detail in the next Section 1.1.1.

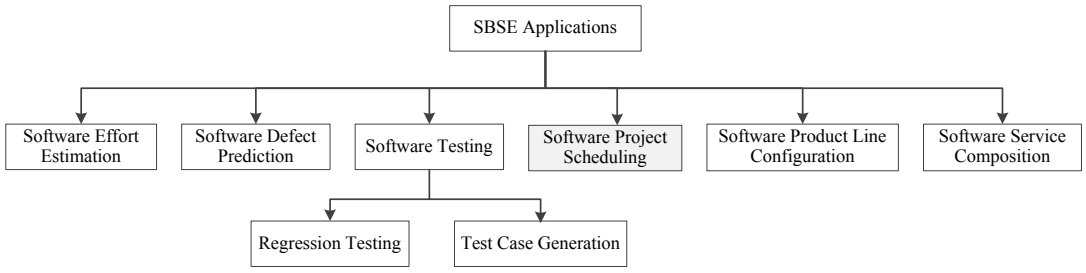



Figure 1.2: Search-based Software Engineering Applications

### A. Software Effort Estimation

The software effort estimation is a major and key part of the software development life cycle. It can be defined as a process of predicting an accurate and realistic amount of effort required to develop or maintain a software project. These estimates for the effort (expressed in terms of person-hours or money) may be used as input to budgets, project plans, iteration plans, investment analyses, pricing processes, and bidding rounds. There is overwhelming evidence towards cost and effort overruns in software projects due to incorrect effort estimates. Therefore, many studies [112, 87, 68] addressed this problem using SBSE approaches in a promising way.

### B. Software Defect Prediction

A reliable and accurate prediction of defect-prone software modules has always been a challenge for both the software industry and academia. In software development process, the cost to find and correct software defects have been the most expensive activity. An accurate defect prediction can affect and reduce the software cost and testing effort. The software testing process can also be improved by focusing on fault-prone modules. Many studies [158, 159, 23] have applied SBSE techniques to solve this problem and results show that they significantly improve the prediction performance.

### C. Software Testing

The software testing is a primary way used to build and maintain the customer's confidence in the software. This utilizes approximately 50%-60% of total software development cost, 40%-50% of total resources, and 30% of total effort [98]. Unfortunately,

software testing is always a time-consuming and costly task [15]. In this regard, studies [162, 101, 148, 102, 174, 176] that have applied SBSE techniques to support the automation of software testing, have resulted in significant cost savings.

#### D. Software Product Lines Configuration

The software product lines (SPLs) [36] is an important software development paradigm. It refers to the collection of similar software products that are configured systematically using common means of production, and reusable modular software components. Software engineers use SPLs to enhance software reusability, minimize software cost, and support software maintenance [81, 94]. During the last decade, there has been significant interest in SPLs. Many researchers found it an interesting problem and sought to solve it using evolutionary approaches [84, 166].

#### E. Software Service Composition

The service oriented computing has a main trend in software engineering. It composes a software application by aggregating different, seamlessly connected services deployed over the Internet according to a given work flow [14]. The service composition is a type of service oriented computing that enables the rapid realization and different functionalities integration as required by the stakeholders. The SBSE approaches [31, 32] have become a promising approach to deal with software service composition while simultaneously optimizing multiple conflicting Quality-of-Service (QoS) objectives, e.g., latency, throughput and cost.

### 1.1.1 Software Project Scheduling

The software project scheduling is a special case of classical project scheduling where the project success (cost, duration) depends on the individuals' intellectual capabilities (skills sets) in the software development team. The uncertainties in the SPS are not primarily due to the involvement of different suppliers' tiers but highly depends on the dynamic events (e.g. employee resignation, employee addition, employee experience). Moreover, the classical project scheduling is static whereas SPS requires frequent project reschedules to make sure that project is on track (cost, schedule). This requires

additional objectives to be considered as robustness and stability.

The software project scheduling (SPS) deals with suitable allocation of employees to tasks in a software project. According to the best practices of software engineering, the total budget and human efforts are key parameters for successful project completion, and must be managed well enough. Therefore, software project scheduling is an important software engineering problem. It is one of those NP-hard combinatorial optimization problems for which evolutionary algorithms (EAs) have been widely applied [137] and proven to help the project managers for near-optimal software project plans. Thus, in SBSE, the SPS is formulated as a search problem with the goal to find the most appropriate solution conforming to some adequacy criteria (e.g., minimizing project cost and duration). Rather than constructing project schedules, SBSE simply searches for them. Since, the project cost and duration are in conflict with each other; as when one is reduced, usually the other increases. Therefore, the goal of optimization process is to find a set of solutions that represent a trade-off between both objectives.

Most of the existing studies address this problem by considering only static and deterministic scenarios, where no disruption occurs [5, 27, 28, 30, 111]. However, real-world software projects are subject to environmental changes, e.g., employee resignation, new employee addition, task effort uncertainty and consideration of important human factors, e.g., employee experience and skills. Major challenges are to deal with the SPS problem under dynamic and uncertain environment to avoid any infeasible schedules. In this thesis, novel approaches based on evolutionary algorithm are presented to address these challenges, with the aim to generate an effective software project plan in a dynamic environment.

## 1.2 Motivation

In recent years, the rapid growth in software industry has resulted in the emergence of highly competitive market where success heavily depends on the faster but within budget completion of software projects [118]. Therefore, it has been argued that a successful project should fulfil the following criteria [92]: i) completed within the planned

time and budget, ii) satisfies project performance measures, iii) fulfils project stakeholder needs and expectations, and iv) completed within the defined and agreed scope. This refers to a software project scheduling (SPS) problem which deals with suitable allocation of employees to software tasks during the whole software project development life cycle [5]. The SPS seeks to assign employees to tasks by taking into account employees' skills and preferences, organizational and legal rules, and other applicable requirements. It is a common problem for most of the organizations. The SPS problem complexity involves assigning the right employee to the right task at the right time. According to [151], a good scheduling system meets essential characteristics as: i) efficient to meet due-dates and production costs and robust enough to handle exceptions, ii) uses constraints knowledge, preferences, and current information about the environment during schedule generation, iii) provides enough flexibility to react to disruptions in an efficient and timely manner, and iv) to increase the efficiency of the system in real time, scheduling decisions are based on actual conditions. In this regard, the main SPS challenges are [121]: evaluating schedule effectiveness, dealing with uncertain dynamic scenarios, accommodating unpredictability, and reducing employees' workload.

Although, the software project scheduling problem has been intensively explored in the literature, but most of the studies focus on static and deterministic scenarios (no disruption occurs during the whole software development life cycle). These may result in performance deterioration and infeasibility when explored to unpredictable scenarios. Hence, the current state-of-the-art research demonstrates a lack of flexibility to effectively cope with dynamic and uncertain events which usually occur during the development of software projects such as, employee resignation, new employee addition, task effort uncertainty, and volatile requirements etc.

Therefore, researchers have to deal with a number of challenges for the SPS problem. Most notably, researchers deal with the static version of SPS problem considering that no uncertain event can trigger. In these studies, real-world dynamic scenarios are ignored. Moreover, they lack the usage of real-world data set and use case studies or some benchmarked problems while evaluating their techniques. Furthermore, several studies have used the same problem formulation as defined by Alba and Chicano [5].

Another challenge is that they mainly deal with two objectives to be optimized usually time and cost where tasks' efforts are known in advance. The scalability of these approaches also needs more validation.

All these real-world challenges shape up the SPS problem as multi-objective dynamic software project scheduling problem under uncertain environments. Consequently, three real-world challenges are modelled as multi-objective (project duration, cost, robustness, and stability) models, subjected to variety of practical constraints, to deal with dynamic events as 'employee turnover', 'new employee addition', and 'employee experience' along with multi-objective evolutionary algorithms as novel heuristic approaches in the light of analysis on existing state-of-the-art algorithms performance.

## 1.3 Research Methodology

This thesis adopts a classical research methodology of Peffers et al. [122] to guide the research. The following five steps describe the adopted process:

- **Problem Identification and Motivation**: The first step is to gain insights into software project scheduling problem under dynamic environment. For this purpose, a survey is conducted that advanced the understanding of the area which identified the room for improvements and challenges. Based on the findings, interests are narrowed towards developing approaches to handle dynamic events that may occur during software project life cycle. Also, it is considered to build a new model for the SPS problem incorporating important human factor that could affect the software project duration and cost significantly.

- **Objectives Definition for a Solution**: The main objective of this thesis is to devise approaches that could deal with different dynamic events that may trigger during software project life cycle. Further, it is aimed at constructing the SPS model that incorporates an important human factor and identify which state-of-the-art algorithm better handles this factor. These objectives are formulated in the form of Research Questions 1-5 in Section 1.4.

- **Design and Development**: A survey is carried out for the SPS problem. The results of survey reveal the existing approaches inadequacies. In this context, metaheuristic techniques are adapted that have shown to be effective for the SPS problem. In particular, the models and approaches are developed to deal with dynamic events with the aim of optimizing project objectives.

- **Demonstration**: To motivate the designed and developed approaches, this thesis uses 18 benchmark problem instances and six real-world data instances.

- **Evaluation**: To compare the performance of proposed approaches against exiting methods, an experimental quantitative evaluation is used. Specially, the performance of the approaches is compared in terms of project objectives (duration, cost, robustness, and stability) using one of best known quality indicator - hypervolume (HV), and project schedules drawn by Gantt chart.

## 1.4 Research Questions

This thesis addresses the following research questions (RQ):

- **RQ1**: Can existing methods deal efficiently with the dynamic events 'employee turnover' and 'new employee addition'? (Chapter 3 and Chapter 4 respectively)

- **RQ2**: Does rescheduling of existing employees delay the project upon employee turnover? What should be the acceptable skill set of new employee being hired in case if project cost and duration is extended? (Chapter 3)

- **RQ3**: Does the improvement on objectives (duration and cost) made by the proposed heuristic to deal with 'new employee addition', compromise other objectives, e.g., project robustness and stability? (Chapter 4)

- **RQ4**: Do the strategies designed into our algorithm handle dynamic events more effectively than state-of-the-art rescheduling methods? (Chapter 4)

- **RQ5**: How can we characterize the notion of 'employee experience' for the SPS problem? How do state-of-the-art algorithms perform on the proposed model?

Do they perform similarly, or are specific algorithms particularly suitable for this model? (Chapter 5)

## 1.5 Contributions

The research described in this thesis contributes to the general software project scheduling problem under a dynamic environment. In particular, the thesis contributes to novel approaches to deal with dynamic events and consideration of important human factors for the SPS problem. Specifically, the main contributions of the thesis are listed below:

1. **Conceptual Model and MOEA-based Approach Towards 'Employee Turnover' Dynamic Event**: A model is introduced based on real-world situation to handle the 'employee turnover' dynamic event. This model supports both employee rescheduling and new employee recruitment scenarios. We are the first to check if rescheduling the remaining team a bit can meet the manager's requirements based on the given project budget; if not, then we attempt to provide the manager with a set of options about what kind of person he/she could recruit within the budget. A new realistic way is also devised for the calculation of an employee's proficiency for a task. Moreover, novel multi-objective evolutionary approaches are proposed for: i) employees rescheduling and ii) to help software project managers in finding new employees to fulfil the required missing skills when someone resigns in the middle of software development project. It is also demonstrated how rescheduling affects the project schedule. Furthermore, we also show that how our proposed approach helps the software project manager in recruiting a new employee to avoid any hiring delay.

2. **A Heuristic-based Approach for 'New Employee Addition' Dynamic Event**: A novel algorithm, h_NEA, is proposed to deal with 'new employee addition' dynamic event. h_NEA is based on a heuristic approach to deal with the SPS problem. Specifically, with an aim to minimize project cost and duration, we design heuristic strategies for each project objective i.e. duration and cost. The duration heuristic will optimize the project duration by allocating a new

employee on critical tasks while the cost heuristic tries to minimize the cost by employee replacement based on specific constraints. The employee replacement will only occur if there is significant difference in cost. Furthermore, it is also investigated how our proposed heuristic affects other project objectives.

3. **SPS Model based on Human Aspects**: A new SPS model is proposed including an important human factor i.e. employee experience. This added parameter makes the SPS model more realistic. Based on this model, two project objectives are optimized as duration and cost. Moreover, the cost objective is defined according to real-world scenario. It is also demonstrated how the proposed model significantly reduces project cost and duration as compared to the state-of-the-art model. Furthermore, systematic experiments are carried out to compare the performance of six state-of-the-art algorithms on the proposed model, using 18 benchmark and six real-world data instances. The results reveal that one algorithm is particularly suitable for the proposed model - it can generate quality project plans which strike a good balance between project cost and duration.

The above three contributions represent a significant advancement in the SPS problem under a dynamic environment which should provide help in both industry and academia for algorithm development and problem solving.

## 1.6 Thesis Structure

This section presents structure of remainder of the thesis as outlined below:

**Chapter 2** presents the necessary background material for the thesis. This chapter begins with the basic idea of Software Project Management and then, defines the software project scheduling problem in detail. Besides basic concepts about multi-objective optimisation, this chapter introduces evolutionary multi-objective optimisation techniques for addressing the SPS problem, decision maker details, and performance indicators. Finally, we review related work under both static and dynamic contexts.

**Chapter 3** starts with the background and motivation of the work. Then, a conceptual model along with multi-objective evolutionary approach is presented to deal

with the 'employee turnover' dynamic event. We also describe the employee's proficiency calculation in a more realistic way followed by project objectives and constraints. Finally, we investigate the proposed approaches as well as their implementation.

**Chapter 4** begins with the motivation of our work. Then, a heuristic-based approach (h_NEA) to handle the 'new employee addition' dynamic event is presented. Next, we formulate the SPS problem using a mathematical model which defines project objectives and constraints. Finally, further investigation is provided to empirically verify the performance of h_NEA using a comparative study based on project schedules and a quality indicator namely, hypervolume (HV).

**Chapter 5** proposes a model including the 'employee experience' factor for the SPS problem. This chapter starts with the motivation of this work and then details the model based on the SPS problem main attributes, objective functions and constraints. Next, experimental results are shown of the proposed model in comparison with state-of-the-art model using a baseline algorithm, and finally a further investigation of the performance of six state-of-the-art algorithms on proposed model is done.

**Chapter 6** concludes the thesis and looks at how it has contributed to the area of software project scheduling in a dynamic context. Furthermore, several directions of future research are presented.

## 1.7 Publications

The research presented in this thesis is based on following papers.

- N. Nigar. Model-based dynamic software project scheduling. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 1042-1045, 2017.
- Natasha Nigar, Miqing Li and Xin Yao, "Multi-objective dynamic software project scheduling : Taking on an employee addition", submitted to Evo* (EuroGP 24th European Conference on Genetic Programming), 2021.
- Natasha Nigar, Miqing Li and Xin Yao, "Multi-objective Software Project Scheduling: Turn around the Employee Turnover", 2020, in preparation for submission.

# Chapter 2

# Literature Review

The aim of this chapter is to summarize the literature on: i) software project scheduling (SPS) problem under both static and dynamic contexts and ii) search-based software engineering (SBSE) approaches to solve this problem. The SBSE is a vast topic; hence, in this chapter focus is primarily on the topics relevant to our work.

The remainder of this chapter is structured as follows. In Section 2.1, we give background information covering the SPS problem structure, evolutionary multi-objective optimisation basic concepts and techniques with particular focus on the algorithms used in this thesis for comparison. Section 2.2 reviews the related work regarding the SPS problem under both static and dynamic contexts. Finally, Section 2.3 summarises the chapter.

## 2.1 Background

### 2.1.1 Defining the Problem

Software project management (SPM) is a sub-discipline of project management. It involves scheduling, planning, monitoring, and controlling of software projects to achieve specific objectives, while satisfying a variety of constraints. This thesis mainly deals with the software project scheduling (SPS) problem under SPM. The main focus of solving this problem is to create a schedule for a project with minimal duration and cost by appropriate allocation of employees to tasks. Generally, we can define a project

as the planned set of interrelated tasks that are executed within specified time and cost to achieve one or more specific objectives [88]. A software project is a complete procedure of software development within a specified period of time to achieve the desired software. A project can be characterized as: i) well-defined tasks, ii) it has a unique and distinct goal, iii) it does not involve any routine activity or day-to-day operations, iv) every project is associated with a start time and end time, v) when the project goal is achieved, it ends; hence, it is considered as a temporary phase in organization's life time, and vi) adequate resources are needed for a project in terms of time, manpower, finance, and material.

Based on above definitions, employees and tasks may be distinguished as two major elements of the SPS problem as shown in Figure 2.1.1.

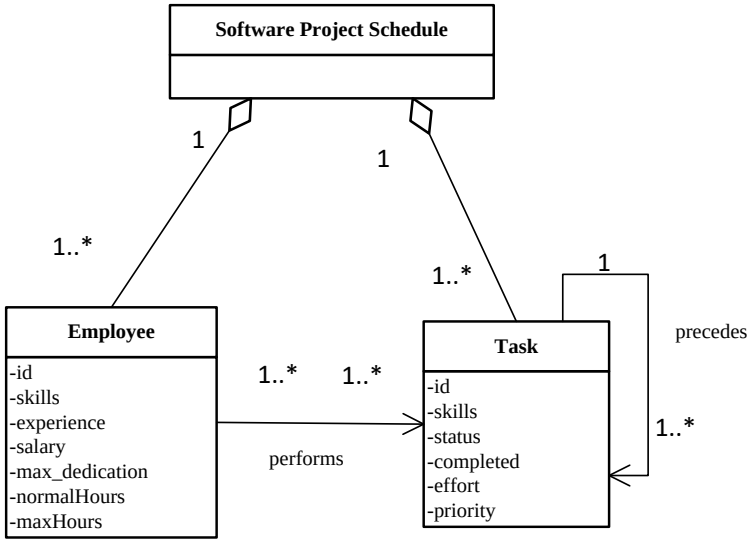



Figure 2.1: Software Project Scheduling Main Entities

To elaborate this, let us suppose a software project comprising of five tasks $T = \{t_1, t_2, t_3, t_4, t_5\}$ and seven employees $E = \{e_1, e_2, e_3, ....., e_7\}$ with employee's skills set $SK = \{s_1, s_2, s_3, s_4, s_5\}$. The skill of an employee is denoted as $e^{skill} \subseteq SK$, the monthly salary as $e^{salary}$ and maximum dedication to the project with $e^{maxded}$. Each skill is associated with a proficiency value in the interval (0,5] [139]. The proficiency value of '0' implies that employee is not proficient for that skill, and a value of '5'

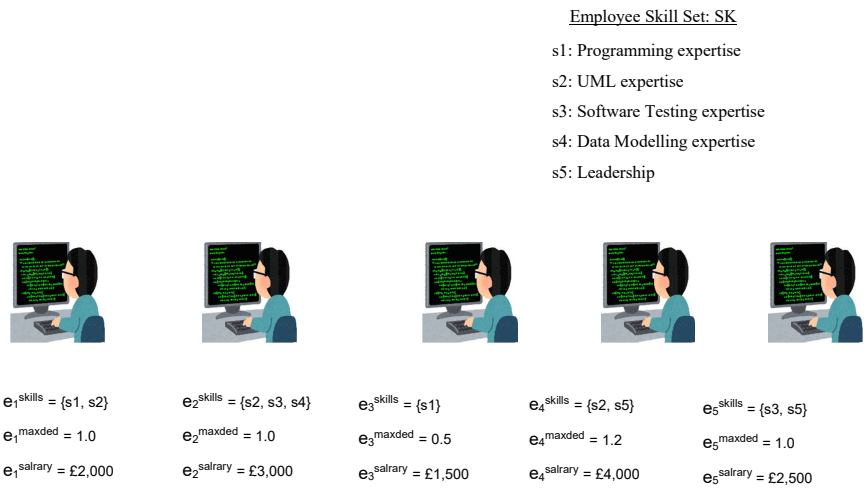



Figure 2.2: Example of Software Project Employees

indicates that employee is fully master in a skill. Salary is expressed in fictitious currency units, while the maximum dedication is defined as the ratio of amount of hours dedicated to the project divided by full length of the employee's working day. Here, salary and maximum dedication are real numbers. A simplified scenario example is given in Figure 2.2. Employee $e_1$ who earns £2,000 each month, is a programming ($s_1$) and UML ($s_2$) expert. His/her colleague employee besides UML ($s_2$) expertise, is also a software tester ($s_3$) and data modeller ($s_4$). These employees ($e_1, e_2$) and employee $e_5$ can spend all their day on the project as they possess maximum dedication equal to one; however, this does not necessarily mean that they spend all of their time. On the other hand, employee $e_3$ is a programmer ($s_1$) and can only dedicate half of his/her working day developing the software application. This may be due to the several reasons, perhaps the employee contract is part-time; or he/she has to look after some administrative tasks as part of their dedication. Employee $e_4$ can work overtime ($e^{maxHours}$) with his/her maximum dedication greater than one ($e^{maxded} = 1.2$). This implies that he/she can work on the project up to 20% more than in a normal working day ($e^{nhours}$).

Moreover, each employee has an experience attribute ($e^{exp}$) in the SPS problem which evolves with time. Employee experience has become an important trend in both human resource and businesses, therefore, we also consider this new attribute into the

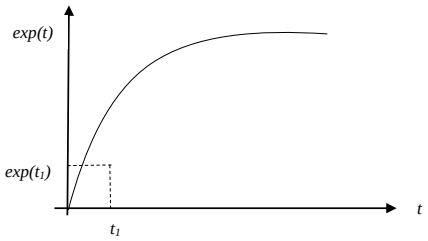



Figure 2.3: An Illustration of Evolving Employee Experience over Time '$t$'

SPS problem. An illustration of evolving employee experience is given in Figure 2.3.

Now the modelling of software tasks is explained. Each task is associated with some skills denoted as $t^{skills}$ and an effort $t^{effort}$ expressed in person-month (PM). The tasks are executed based on a Task Precedence Graph (TPG). This indicates which tasks must be completed before a new task begins. TPG is an acyclic directed graph G(T,A) to represent the dependency between tasks. In TPG, the set of nodes represents the set of tasks T. The precedence relation among the tasks is denoted by set of arcs A. $t^{completed}$ is a binary variable indicating that whether a task has been finished at time $t$. $t^{completed} = 1$ shows that task is still in execution whereas $t^{completed} = 0$ represents that task has been completed. Another binary variable $t^{status}$ represents task's availability that whether a task is active at time $t$ or it has been cancelled/suspended, e.g., due to some missing skill. If $t^{status} = 1$, it shows that task is available whereas $t^{status} = 0$ represents that task has been cancelled/suspended. $t^{prio}$ represents task's priority level i.e. very high, high, or medium. Another variable is $e_{ij}^{prof}$, it indicates employee $e_i$ proficiency level for performing a task $t_j$. The employee proficiency level for a given task is computed and explained in Chapter 3.

The Figure 2.4 shows all software project tasks considered in the example. Task $t_1$ requires UML expertise (skill $s_2$) to prepare the design model document in order to be used later by the employees in completing the subsequent tasks, and it needs 3 person-months to be completed. In the same figure, we show the dependency between tasks. The last task $t_7$ (User manual creation) cannot be instantiated until all previous tasks are completed.

The aim is to minimize the project duration, cost, robustness, and stability objectives for a software project. For the SPS problem, there are also number of practical

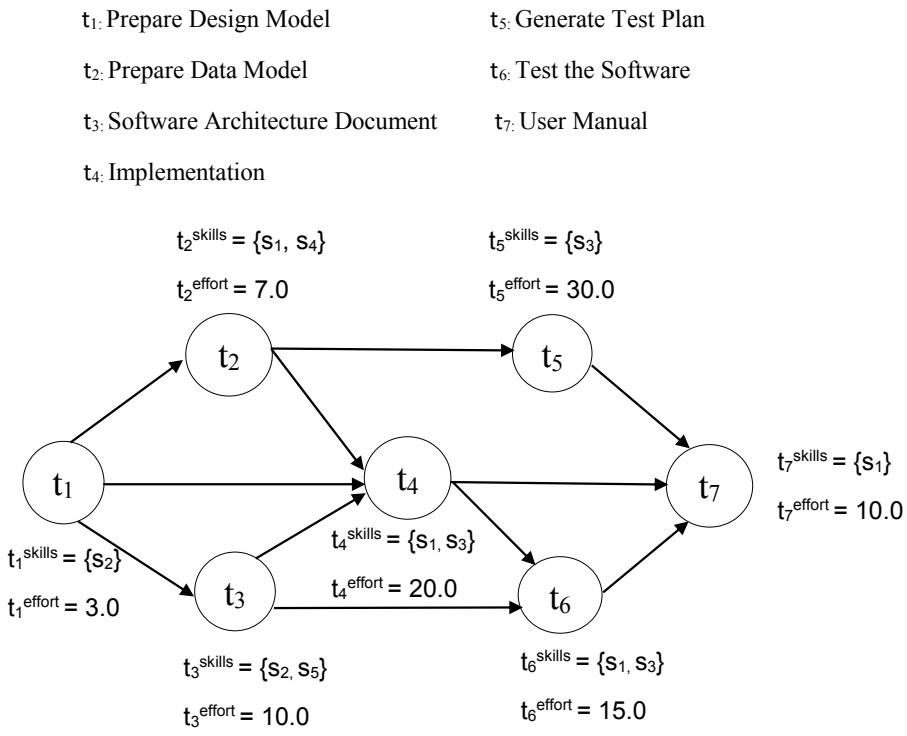



Figure 2.4: Example of Task Precedence Graph

constraints, below the first three are hard constraints and the last is a soft one.

i. Each task must be performed by at least one available employee.

ii. Each task skills must be fulfilled by all the employees allocated on that task.

iii. No employee should overwork. This may produce the final product with lower quality and, possibly, lead to the correction of or re-development of the erroneous parts, resulting in an increase of project duration.

iv. Task headcount is a soft constraint for the SPS problem. It defines that each task should have a maximum number of employees allocated to it. More employees assigned on a task may increase communication overhead which may result in low productivity.

**Solution to the SPS problem.** After presenting the employees and tasks as two major elements of SPS problem. The solution to the SPS problem can be represented

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ |
|---|---|---|---|---|---|---|---|
| $e_1$ | 1.00 | 0.50 | 0.50 | 0.00 | 0.00 | 1.00 | 0.87 |
| $e_2$ | 0.50 | 1.00 | 1.00 | 0.33 | 0.50 | 0.50 | 0.00 |
| $e_3$ | 0.00 | 0.70 | 0.00 | 0.34 | 0.00 | 0.33 | 0.33 |
| $e_4$ | 0.67 | 0.00 | 0.87 | 0.00 | 0.00 | 0.00 | 0.00 |
| $e_5$ | 0.00 | 0.00 | 1.00 | 0.00 | 0.90 | 0.00 | 0.00 |

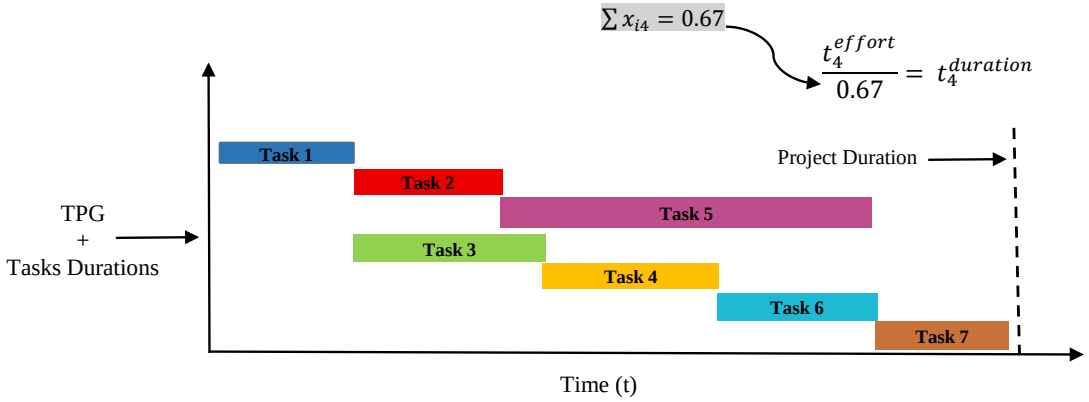
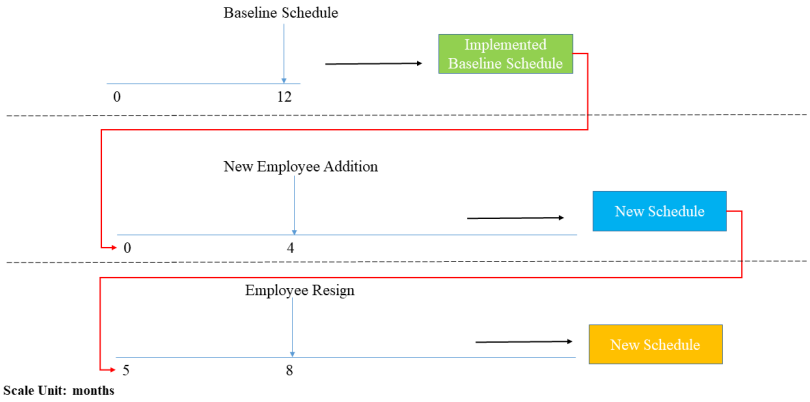



Figure 2.5: Tentative Solution to the SPS problem

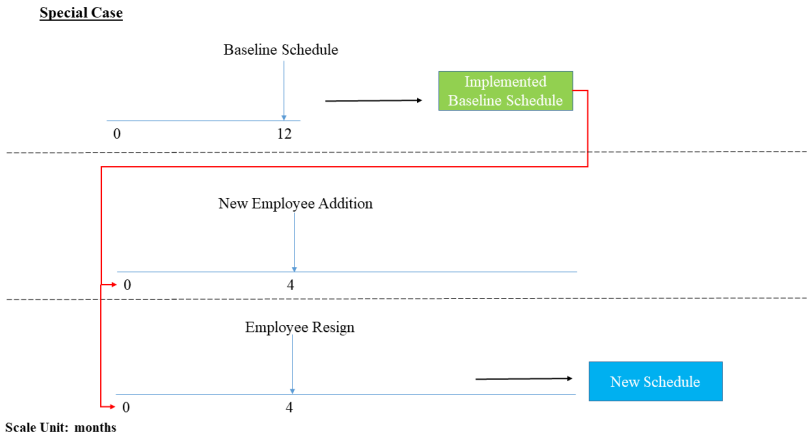
by a dedication matrix $X = (x_{ij})$ of size $|E|$ x $|T|$ where $x_{ij} \geq 0$. The element $|E|$ represents number of employees, $|T|$ denotes number of tasks, and $x_{ij}$ represents the degree of dedication of an employee $e_i$ to task $t_j$. For example, an employee $e_i$ is allocated on a task $t_j$ with dedication degree of '1' means that he/she spends his/her all normal working hours of a day to that task. $x_{ij} = 0$ means that employee will not perform that task. This information is used to calculate the duration and starting and finishing time of each task. According to the TPG and dedication matrix, we can also draw Gantt chart of the project as shown in Figure 2.5. A task's duration is calculated according to the formulation in [5]. Once the duration of a project is calculated we can calculate project cost using the dedication matrix and employees' salary. An example of a tentative Gantt chart is also given.

**Scheduling Approaches.** The focus of this thesis is to solve the SPS problem in a dynamic and uncertain environment. For this purpose, there could be two possible approaches to deal with this problem: i) Offline Approach and ii) Online Approach, which are presented below:

***i. Offline Approach***. This approach is close to real-world situations. An ini-




(a) Offline Approach




(b) Offline Approach - Special Case

Figure 2.6: An Offline Scheduling Approach

tial/baseline schedule is generated and implemented. If a dynamic event occurs, then based on the updated information (e.g., available employees, tasks, updated task precedence graph (TPG), and remaining tasks' efforts) a new schedule is regenerated (Figure 2.6). There is also a special scenario that if two dynamic events trigger simultaneously, then, this approach is also able to handle such a situation. As dynamic events are unpredictable during the development of a software project, therefore, it is also possible that an event may never trigger. Hence, based on these facts we have adopted this approach to deal with the SPS problem.

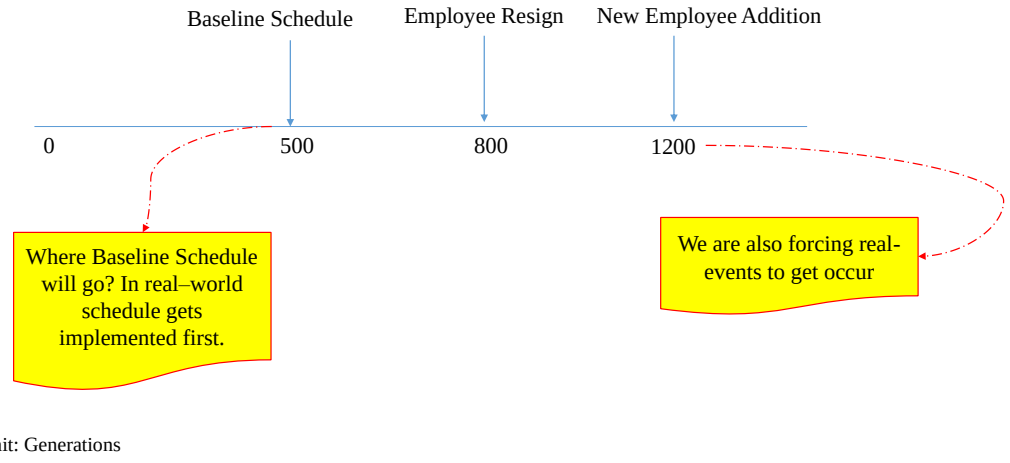



Figure 2.7: An Online Scheduling Approach

***ii. Online Approach***. Another possible strategy could be an online approach (Figure 2.7) by employing an evolutionary algorithm where dynamic events trigger after a certain number of generations. It is more relevant to a dynamic optimization method. However, in the real world, a schedule is implemented first and then it deals with any triggered dynamic event.

**General Framework for SPS problem**. At initial time $t_0$, when a project starts, a robust schedule for all the tasks and employees is generated considering project objectives (duration, cost, and robustness). Let's say at time $t^{'}(t^{'} > t_0)$, also called scheduling point, an unpredictable event occurs. At this time, a new schedule is regenerated considering all the updated and current project information, which contains the set of available employees, the set of available tasks with their remaining estimated task efforts, and updated task precedence graph (TPG). This framework is based on the offline approach as mentioned above. At scheduling point $t^{'}$, the stability objective is also considered. Figure 2.8 also depicts this procedure.

### 2.1.2 Evolutionary Multi-objective Optimization

In this section, we introduce the key parts in evolutionary multi-objective Optimization (EMO), including introduction to multi-objective Optimization and mainstream EMO methods (used in this thesis) in the following sub-subsections.

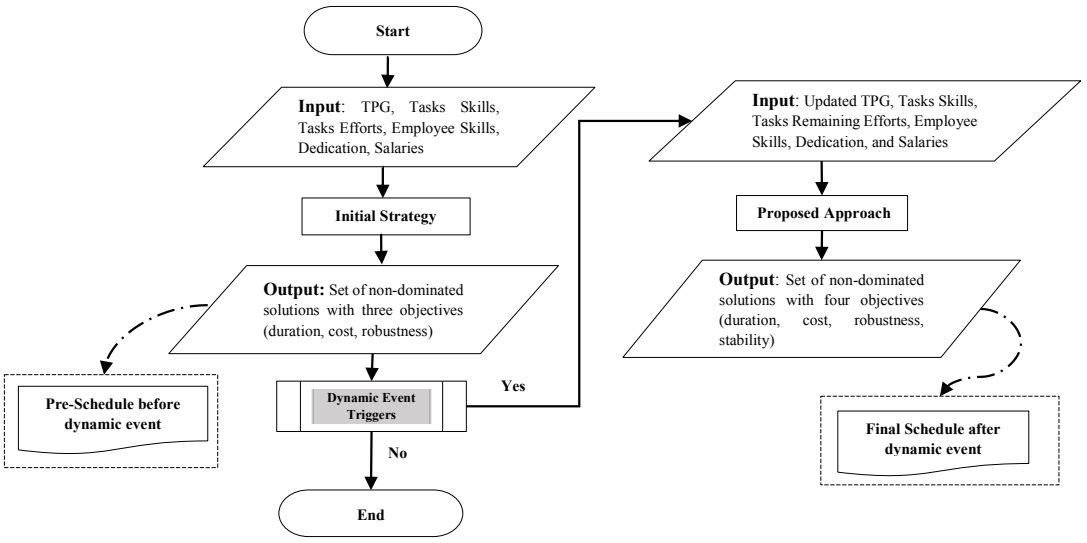



Figure 2.8: A General Framework for SPS Problem

#### 2.1.2.1 Multi-objective Optimization

Since the majority of real-world optimization problems involve multiple conflicting objectives along with constraints, multi-objective optimization is an integral part of optimization activities and has a tremendous practical importance. It involves more than one objective function to be optimized simultaneously. For example, in buying a car, minimizing cost while maximizing comfort, and maximizing performance whilst minimizing fuel consumption and pollutant emissions of a vehicle are examples of multi-objective optimization problems involving two and three objectives, respectively. In general, we define a multi-objective optimisation problem (MOP) as an optimization problem that involves $m$ objective functions, $n$ decision variables, and $K$ constraints. A MOP involving more than one conflicting objective to be optimized simultaneously, is formulated in the following form:

$$\begin{aligned} \text{Minimize} \quad & f_i(\mathbf{x}), i = 1, 2, ..., m \\ \text{Subject to} \quad & g_k(\mathbf{x}) = 0, k = 1, 2, ..., K \\ & LB_j \leq x_j \leq UB_j, j = 1, 2, ..., n \end{aligned} \tag{2.1}$$

where $f$ represents $m$ objectives to be minimized and $\mathbf{x}$ is a vector of $n$ decision variables: $\mathbf{x} = (x_1, x_2, ..., x_n)$, $\mathbf{x} \in \mathbb{R}^n$. Further, the variable bounds restrict a decision variable $x_j$ within the range of $[LB_j, UB_j]$. Since there exists a conflict between the optimization objectives $\mathbf{f_1}(\mathbf{x}), \mathbf{f_2}(\mathbf{x}), ..., \mathbf{f_m}(\mathbf{x})$ in an MOP as formulated above, it is impossible to find one single solution that optimizes all objectives simultaneously. We are particularly interested in the performance of members of the set of feasible solutions. A solution is called feasible (denoted as $\mathbf{x} \in \mathbb{R}_f^n$) if it satisfies all constraints. In the following, some underlying concepts in multi-objective optimisation are introduced.

**Definition 2.1.1** (Pareto dominance)**.** *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *are two decision variables,* $\mathbf{x}$ *is said to Pareto dominate* $\mathbf{y}$*, denoted as* $\mathbf{x} \prec \mathbf{y}$*, such that:*

$$\begin{aligned} &\forall i \in (1, 2, ..., m) : f_i(\mathbf{x}) \leq f_i(\mathbf{y}) \wedge \\ &\exists j \in (1, 2, ..., m) : f_j(\mathbf{x}) < f_j(\mathbf{y}) \end{aligned} \tag{2.2}$$

In MOP, a solution is known as *Pareto optimal* if that is not dominated by any other existing solution. Such solutions are attributed with the fact that improvement to one objective comes at the expense of another objective. In the decision space, the union of Pareto optimal solutions is called the *Pareto set (PS)*, and the corresponding objective vectors set is known as the *Pareto front (PF)*. The definitions for these concepts are given below.

**Definition 2.1.2** (Pareto optimality)**.** *A solution* $\mathbf{x} \in \mathbb{R}_f^n$ *is said to be Pareto optimal if and only if* $\nexists \mathbf{y} \in \mathbb{R}_f^n$, $\mathbf{y} \prec \mathbf{x}$.

**Definition 2.1.3** (Pareto set)**.** *The set of all Pareto optimal solutions is defined as the Pareto set (PS), namely,* $PS = \{\mathbf{x} \in \mathbb{R}_f^n | \nexists \mathbf{y} \in \mathbb{R}_f^n, \mathbf{y} \prec \mathbf{x}\}$.

**Definition 2.1.4** (Pareto front)**.** *The set of all objective vectors corresponding to the solutions in PS is defined as the Pareto front (PF), namely,* $PF = \{(f_1(\mathbf{x}), ..., f_m(\mathbf{x})) : \mathbf{x} \in PS\}$.

Since the Pareto optimal solutions size might not be finite, therefore, it is usually not feasible to obtain the whole Pareto front. In real examples, an approximation of the

Pareto front is desired that is 'close to' the actual Pareto front and satisfies some further desirable criteria (see below). For the decision maker (DM), either this information can be used to set the preferences for searching and finding a satisfied solution; or the DM can select one solution from the approximation set as the final solution.

### A. Convergence and Diversity

Convergence means that non-dominated solutions maintained by the optimization technique should be as close as possible to the exact Pareto front [50, 52]. Diversity means that solutions are uniformly spread. It indicates good exploration of search space and no regions are left unexplored [50, 52].

### B. Exploration and Exploitation

Exploration and exploitation are two key components of nature-inspired optimization algorithms. These algorithms can also be analysed from the ways they explore the search space. Exploitation (also known as intensification) [16] uses any information obtained from the problem of interest and helps in generating new solutions that are better than existing solutions. However, this process and information are typically local; hence, it is ideally suited for local search. For example, hill climbing is a local search method in which derivative information is used to guide the search procedure. Exploitation leads to very high convergence rates. Its disadvantage is that it can get stuck in a local optimum because the final solution point largely depends on the starting point.

In contrast, exploration explores the search space more efficiently on a global scale. As a result, solutions with enough diversity are generated and are far from the current solutions. The exploration is less likely to get stuck in a local optimum, and the global optimality can be more accessible. However, it has some disadvantages as well, e.g., i) slow convergence and ii) the waste of some computational efforts because many new solutions can be far from the global optimality.

#### 2.1.2.2 Evolutionary Computation

The use of evolutionary computation (EC) was inspired by the Darwinian model of evolution and biological theory [47] to solve the engineering and science problems. EC uses Darwin concepts and principles of evolution and natural selection to solve complex problems.

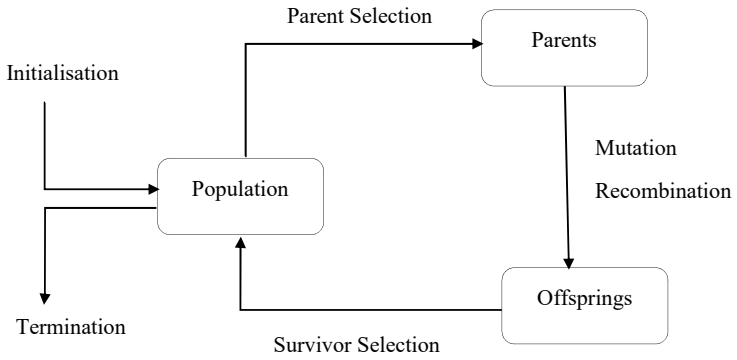
**A. Evolutionary Algorithms**

Evolutionary algorithms (EAs) are a subset of evolutionary computation. The computations in EAs are implemented based on mechanisms inspired by biological evolutions. Over the past two decades, EAs have been used widely to solve MOPs, in many diverse fields e.g. engineering, economics, chemistry, physics, biology, marketing, operations research, and social sciences [49, 38, 37, 177, 123]. This can be attributed to two distinct features of EAs: i) EAs are stochastic and are not tied to specific problems. They have capability to handle large and highly complex search spaces and ii) EAs are population-based, i.e., they process many candidate solutions simultaneously. They can achieve an approximation of the Pareto front for the problem. In this way, each obtained solution represents a particular trade-off amongst the objectives. In general, with respect to its solution set, an EMO algorithm has the following goals:

i. minimize the distance to the Pareto front, also known as *convergence*.

ii. maximize the distribution over the Pareto front, also called *diversity*.

For a given problem, possible solutions are known as 'individuals'. Each individual has a fitness function. Individuals who have higher fitness values prove themselves as better quality solutions for the given problem. Genetic operators (i.e. crossover and mutation) are applied on each individual and it is encoded with chromosomes (which is an abstract representation). Individuals with higher fitness values are selected during the selection process and then offspring are produced by applying crossover and mutation operators, these are called 'candidate solutions'. The new population comprises of the candidate solutions is called a new generation. Over generations, better solutions are evolved gradually under the force of evolutionary pressure. To summarize,




(a) As Flow Chart

```
BEGIN
   INITIALISE population with random candidate solutions;
   EVALUATE each candidate;
   REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
      1 SELECT parents;
      2 RECOMBINE pairs of parents;
      3 MUTATE the resulting offspring;
      4 EVALUATE new candidates;
      5 SELECT individuals for the next generation;
END
```

(b) As pseudo-code

Figure 2.9: The general scheme of an Evolutionary Algorithm

a general scheme for an evolutionary process is given in Figure 2.9 as a diagram and in a pseudo-code fashion. Notably, this scheme falls in the category of generate-and-test algorithms. The search process is driven by variation and selection processes and the evaluation function represents a heuristic estimation of solution quality. Next, different EAs are presented which are used in this thesis.

### i. Non-dominated Sorting GA-II (NSGA-II)

The NSGA-II was introduced by Deb et al. [54] which is an improved version (better sorting) of a popular non-domination based genetic algorithm (NSGA) [147] for multi-objective optimization. It is composed of two principal parts: a fast non-dominated sorting solution and the preservation of the solution's diversity. It incorporates an elitism approach to create a diverse Pareto-optimal front and no sharing parameter

needs to be chosen a priori.

The replacement strategy in NSGA-II is based on Pareto-dominance and crowding distance selection. Since large neighbourhood values allow better diversity in the population, it measures how close a solution is to its neighbours. Let's say $P$ represents a population of solutions, $O$ denotes the population of offspring solutions, and $N$ denotes maximum population size. The surviving solutions are selected from $P \cup O$ by taking non-dominated solutions to compose $P'$ while $|P| < N$, it adds dominated solutions according to the crowding distance values. This crowded comparison operator makes the NSGA-II considerably faster than its predecessor with good results.

Besides its capabilities, NSGA-II also has some disadvantages: i) it has large storage space of size $O(N^2)$, ii) non-dominated sorting of $2N$ size is performed, iii) crowded comparison can restrict the convergence, and iv) NSGA-II shows poor exploration power resulting in small population size.

#### ii. Non-dominated Sorting GA-III (NSGA-III)

NSGA-III [51] is an improved version of NSGA-II and is more suitable to deal with many objectives (more than three) problems. The basic framework of the NSGA-III is similar to its predecessor (NSGA-II). NSGA-III solves the multi-objective optimization problem (MOP) more efficiently by changing the selection mechanism of NSGA-II. Particularly, the main difference is that NSGA-III emphasis on population members that are non-dominated and crowding distance selection is based on well-distributed and adaptive reference points. The main purpose of these reference points is to maintain the diversity of population members and also enable the NSGA-III to deal with differently scaled objective values in a good manner for MOP.

#### iii. Bi-Criterion Evolution (BCE) Algorithm

The BCE [104] is an MOEA-based algorithm. Its framework is composed of two parts: pareto criterion (PC) evolution and non-pareto criterion (NPC) evolution to deal with the MOPs. These two parts work collaboratively, attempting to use their strengths to facilitate each other's evolution. In the framework, the two populations communicate constantly and fully share and compare their information in a generational manner.

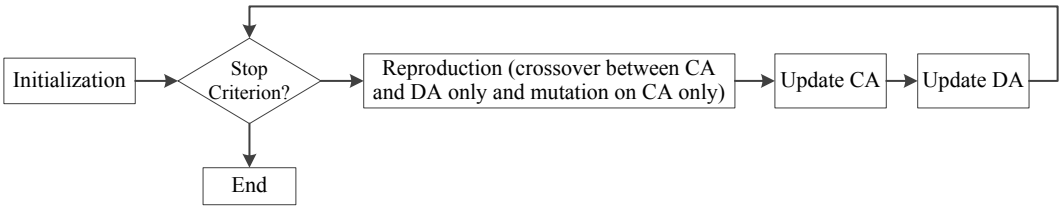



Figure 2.10: A General flow chart of Two_Arch2 Algorithm

NPC evolution drives the PC evolution forward while PC evolution compensates for the possible diversity loss of the NPC evolution. An individual newly produced in one population is tested and applied in the other population. The current status of the NPC evolution is reflected by information comparison of the two populations, thus, making the search more focused on some undeveloped but promising regions.

#### iv. Improved Two_Arch Algorithm (Two_Arch2)

Two_Arch2 [160] is a many objective algorithm comprising of two archives focusing on convergence and diversity separately. The non-dominated solution set is divided into convergence archive (CA) and diversity archive (DA) (Figure 2.10). CA causes the population to converge to the true Pareto front at a fast speed while the goal of DA is to add more diversity to the population in a high-dimensional objective space. The union of CA and DA can be used as parent population for the reproduction step (crossover and mutation). Two_Arch2 is a hybrid multi-objective evolutionary algorithm (MOEA) because CA and DA are updated independently by different dominance relations.

### B. $\epsilon$-MOEA

$\epsilon$-MOEA [53] is a classic $\epsilon$-dominance based algorithm (where $\epsilon$-dominance represents dominance of solutions based on some appropriate $\epsilon$-value). It is a steady-state MOEA with an elitist approach and emphasizes on non-dominated solutions. It uses archiving/selection strategies that guarantee both convergence and diversity. Progress towards the Pareto-optimal set and covering of the whole range of the non-dominated solutions are made at the same time. The use of $\epsilon$-dominance allows the decision-maker to control the resolution of the Pareto set approximation by choosing an appropriate

$\epsilon$-value, thus, making the algorithms practical.

The size of the final archive set of $\epsilon$-MOEA could be completely different in several runs. As an example, the archive set could be only one individual for some runs, yet for others, the archive set may span over 1500 individuals. Another major issue for $\epsilon$-dominance based methods is the determination of suitable $\epsilon$ value, when considering the large number of objectives [105]. Moreover, EMO algorithms based on $\epsilon$-dominance, e.g., $\epsilon$-MOEA, show some sensitivity to the shape of the Pareto front and are not suitable to find the boundary individuals of a population. [82, 103, 171].

### C. Swarm Intelligence

Swarm intelligence (SI) is the collective behaviour of decentralized and self-organized systems, either natural or artificial. Gerardo Beni and Jing Wang introduced this expression in 1989, in the context of cellular robotic systems [12]. It can be categorized in the following types:

#### a. Particle Swarm Optimization

Particle swarm optimization (PSO) is an evolutionary search based metaheuristic developed by Kennedy and Eberhart [91] in 1995. It has emerged as a population-based optimization method which simulates social behaviour exhibited by bird flocks while searching for food. Another characteristic is its inherent ability to search large spaces with few hypotheses for potential candidate solutions. Many continuous non-linear multi-objective constrained problems have been solved successfully as it is problem independent and requires minimum knowledge. In PSO, birds are referred as particles and swarms as particle populations. Each particle in the search space is represented by a position and a velocity which is used by particles to redirect their current positions as a function of cognitive and social elements. These refer to particles' memory on personal best (pBest) and global best (gBest) solutions respectively. Many variations of this standard PSO have been developed for more adaptability to conflicting objectives and fast convergence as multi-objective PSO with pareto solution. It results not only in one best solution but a set of alternative solutions where no one is completely better than others. For the optimization problems where the search space comprises discrete

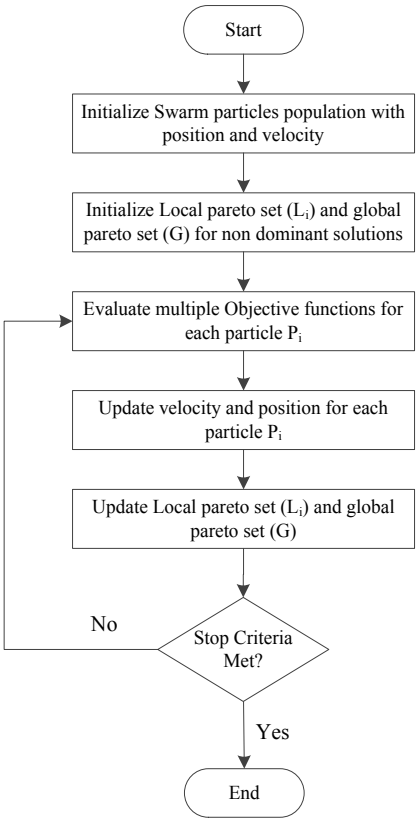



Figure 2.11: A General flow chart of PSO Algorithm

featuring variables, Kennedy and Eberhart (1997) developed a discrete version of PSO. The discrete PSO differs from the continuous PSO in such a way that it is composed of binary variables and its velocity is transformed into probabilities to take either 0 or 1. Unlike evolutionary algorithms, PSO does not use a selection operator; typically, all individuals in a population survive from the beginning of a trial until the end. Individuals' interactions result in iterative improvement of the quality of solutions over generations. The main advantage of such an approach is that it avoids the problem of local minima. A general flow chat for PSO algorithm is given in Figure 2.11. Two different variants of PSO used in this thesis are explained below:

**i. Improved PSO using crowding, mutation and $\epsilon$-dominance concepts (OMOPSO)**

OMOPSO [142] is a multi-objective particle swarm optimization (MOPSO) algorithm which includes an external archive to filter out leader solutions, and uses mutation operators to accelerate the convergence of swarm. The external archive of OMOPSO is based on the crowding distance from NSGA-II and binary tournament is used for selecting one leader for each particle. Moreover, it incorporates a PSO formulation depending on inertia weight to update the velocities at each generation. It also has an archive while applying the concept of $\epsilon$-dominance to store the best solutions found during the search process.

#### ii. Speed-constrained multi-objective PSO (SMPSO)

SMPSO [117] is also a MOPSO which incorporates a velocity constriction mechanism with the aim of enhancing the search capability of the technique. It has also been attributed with the use of polynomial mutation as a turbulence factor. To store the non-dominated solutions found during the search, an external archive is also utilized.

### b. Ant Colony Optimization

Ant colony optimization (ACO) was introduced by Dorigo in 1992 [57]. This method in swarm intelligence was inspired by the foraging behaviour of some ant species, and has been applied to many combinatorial optimization problems, e.g., scheduling problems and vehicle routing problems. Its most popular variants are: elitist ant system, max-min ant system (MMAS), ant colony system, and continuous orthogonal ant colony (COAC). These have been applied to solve the dynamic problems in real variables, stochastic problems, multi-targets and parallel implementations. For the the travelling salesman problem, ACO has also been used to produce near-optimal solutions. ACO has advantage over GA and simulated annealing for the same problem in which the graph may change dynamically. In such case, it can be run continuously and adapts to changes in real time. It is also used due to two main advantages, firstly, the use of a step-by-step construction process to build solutions. In this way, it can build feasible solutions efficiently. Secondly, the step-by-step construction process allows us to use some heuristics to guide the search, whereas in traditional GA and PSO it is difficult to use such heuristic information. The ACO is characterized as a distributed, stochastic

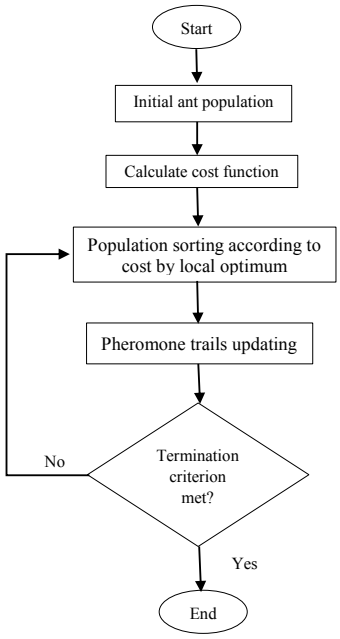



Figure 2.12: A General flow chart of ACO Algorithm

search method based on the indirect communications of artificial ants' colony. When ants travel, they deposit a chemical (called pheromone) on their path to communicate with each other. The pheromone trails serve to construct solutions probabilistically to the problem being solved. The artificial ants reflect their search experience by modifying the pheromone trails during the algorithm's execution. It has been widely applied for solving the discrete optimization problems. The ACO does not give as good optimization results in continuous domains as it gives in discrete domains. A general flow chat for ACO algorithm is given in Figure 2.12.

#### 2.1.2.3 Performance Measures and Selection Procedure

The issue of performance assessment of rapidly developing EMO algorithms has become increasingly important. These are known as quality indicators and used to access the performance of a multi-objective evolutionary algorithm (MOEA). In this section, the quality indicator used in our study is explained. The reason for selecting this quality indicator is that it is more practical for a real-world problem. Moreover, a decision making procedure is also described. EAs provide a set of non-dominated solutions as a

result to solve any NP-hard problem. A project manager may select from these solutions considering best duration, best cost, or best trade-off among all project objectives. In our study, the involvement of a person for taking decisions is not practical. Therefore, an automatic decision making method [138] is used.

### A. Hypervolume

The hypervolume (HV) is one of the most accurate quality indicators [155]. It calculates the volume in objective space (covered by non-dominated solutions set) for problems where all objectives are to be minimized [179]. HV also provides a measure by taking into account the convergence and diversity of the obtained approximation set. One of the distinct features of HV is that it does not need the true Pareto front of the problem. Therefore, it is more suitable in real-world scenarios as compared to other indicators [106].

Let $Q$ represent a set of non-dominated solutions $Q = \{A, B, C, D\}$. For each solution $i \in Q$, a hypercube $v_i$ is constructed with a reference point $W$ and the solution $i$ as the diagonal corners of the hypercube. To calculate the reference point for every problem instance, the maximal values are extracted for all objectives from the Pareto approximations found by all algorithms (used in comparison). These maximal values are used as reference point $W$ for each data instance [155]. Thereafter, a union of all hypercubes is found and hypervolume is calculated as follows:

$$HV = volume(\cup_{i=1}^{|Q|} v_i) \tag{2.3}$$

Algorithms with larger values of HV are desirable. In Figure 2.13, the region enclosed into the discontinuous line (grey shaded area) represents the hypervolume.

### Hypervolume Ratio (HVR)

Hypervolume ratio is the ratio of hypervolume of a solution set $A$ divided by Hypervolume value of Reference Front $B$ as given in (2.4).

$$HV(S, P) = \frac{HV(A)}{HV(B)} \tag{2.4}$$

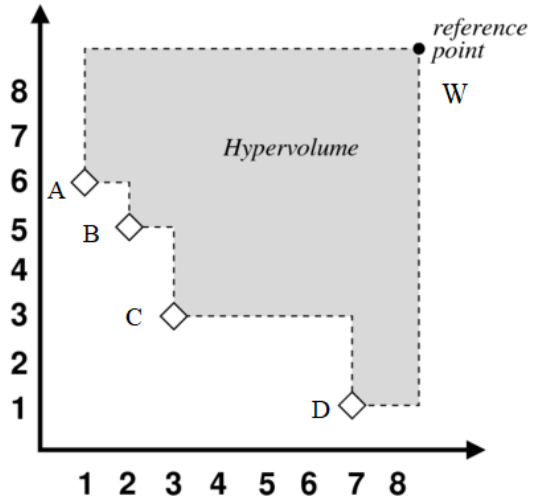



Figure 2.13: Hypervolume enclosed by non-dominated solutions A,B,C,D

### B. Decision Maker

In our study, the decision maker is a small software-based tool which selects a single solution from a set of non-dominated solutions since evolutionary algorithms provide a range of such solutions. In practice, at each scheduling point $t'$, the project manager selects a solution from a set of non-dominated solutions found by the EA; and then the selected schedule is implemented in the project. However, in our study, the involvement of a person for taking decisions is not practical. Thus, an automatic decision making method proposed in [138] is adopted and procedure is briefly explained below:

- Step i (Pairwise Comparison Matrix): In this step, a pairwise comparison matrix is constructed. Let's say if there are $N_o = 4$ objectives to be optimized, then there are $N_o \times (N_o - 1)/2 = 4(4-1)/2 = 6$ comparisons. Then the pairwise comparison matrix $C_1 = (c_{ij})_{N_o \times N_o}$ can be constructed which describes the degree of the preference for one objective versus another.

- Step ii (Weight Vector Estimation): A weight vector $w = (w_i)_{N_o \times 1}$ for multiple objectives is estimated. Here, the logarithmic least squares method [132] is applied. We calculate the geometric mean of each row in the matrix $C_1$, which is then normalized by dividing it by the sum of them.

- Step iii (Objective Values Normalization): Each objective is normalized as:

$$n\_f_i(x) = f_i^{max} - f_i(x)/f_i^{max} - f_i^{min}, i = 1, 2, ...., N_o. \tag{2.5}$$

  where $f_i^{max}$ and $f_i^{min}$ denotes the maximum and minimum objective values among all the non-dominated solutions obtained at the current scheduling point.

- Step iv (Utility Value Calculation): The utility value for each non-dominated solution is found by using the weighted geometric mean of the multiple objective values as follows:

$$U(x) = \prod_{i=1}^{N_o} n\_f_i(x)^{w_i/\sum_i^{N_o} w_i} \tag{2.6}$$

- Step v (Solution selection): The solution which has the maximum utility value is selected as the final schedule.

Notably, the pairwise comparison matrix and the weight vector determined in Steps i and ii are calculated in advanced and not changed during the dynamic process. Only Steps iii, iv, and v are performed at each scheduling point during the project execution.

The following example explains this procedure in detail. Let's say that there are four objectives to be optimized namely, project duration, cost, robustness, and stability. The software project manager gives equal importance to duration and cost objectives, and considers that robustness and stability are of the equal importance. Thus, the pairwise comparison matrix for the four objectives is constructed as follows:

$$C_1 = (c_{ij})_{4\times1} = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1/2 & 1/2 & 1 & 1 \\ 1/2 & 1/2 & 1 & 1 \end{bmatrix}$$

The weight vector $w = (w_i)_{4\times1} = [0.3333, 0.3333, 0.1667, 0.1667]^T$ can be obtained according to the above Step ii. After that each objective is normalized according to Eq.

(2.5), and the utility value for each non-dominated solution can be calculated using Eq. (2.6). Then, the non-dominated solution with the highest utility value is chosen as the final schedule.

## 2.2 Related Work

During last decade, search-based approaches have become a promising way for the SPS problems. These problems are becoming NP-hard due to increasing number of employees and tasks. There are two distinct classes to solve them namely, exact methods and heuristic methods. Heuristic methods have become popular in solving the SPS problems, and include particle swarm optimization (PSO) [39], ant colony optimization (ACO) [57], simulated annealing (SA) [93], Tabu search (TS) [69], and Genetic Algorithms (GAs) [86]. The SPS problem can be further categorized into static and dynamic software project scheduling problems.

### 2.2.1 Static Software Project Scheduling (SSPS)

SSPS considers that no disruption occurs during the whole project development life cycle. For example, no addition or deletion of tasks is permitted. However, a software development project is associated with many uncertain events. In SSPS, if some dynamic scenario occurs, it fails to handle that event. Researchers have tackled the SSPS problem by applying the different metaheuristics techniques which have been elaborated below:

#### 2.2.1.1 SSPS by Genetic Algorithm

Genetic algorithms (GAs) are popular methods to solve the software project scheduling problems. GAs are categorized into stochastic search methods and were introduced by John Holland [86]. Many researchers have used GAs and its variants [43] to study scheduling problems. But no single GA is best fit to solve all the software project scheduling problems. They are tuned and modified to solve a specific problem. Chang et al. [29] made an early effort and applied GAs on a task-based model to provide an

optimal solution. Alba and Chicano [5] used the same task-based model as in [29] and applied GAs for solving many different software project scenarios. They performed in-depth analysis with an instance generator and solved 48 different project scenarios. They also considered few human resource factors but the employees' skills and experience is not distinguished in their model. Moreover, employees' reallocation is not permitted during the whole project duration. Chang et al. [27] proposed project scheduling model (named as software project management net (SPMnet)) and applied GAs to reduce the solution search-space sharply to find near-optimal solutions. As an improvement of their work [30], they used GAs and introduced a 3D matrix representation which specifies work load assignment of each employee for each task at every clock. In their model, task duration is split into small time units. However, a large system instability would be induced as tasks are scheduled separately in different time units.

Xiao et al. [170] also used GAs to allocate human resources to multiple projects by considering various constraints, resource requirements, and value objectives. However, in their model, one human cannot be assigned on more than one activity and overwork of human resources (more than 8 hours per day) is not allowed. Hegazy et al. [80] proposed a model which optimizes cost and dynamic project control. Their model allocates optional construction methods for each task and incorporates an integrated formulation for scheduling, estimating, resource management, and cash-flow analysis. Human factors are not considered much in their work. Their key focus is to give the best combination of construction methods. Chan et al. [25] used GAs for resource scheduling problems. Their objective functions were related to constraints, precedence of task, and resources. They focused more on the algorithmic side in their work, and lack in describing about the motivational and de-motivational factors of projects in scheduling. However, GAs application to SPSP does not provide satisfactory and optimal solution for all project scenarios, and hence, there is a need to deal with these scenarios by means of dynamic scheduling.

#### 2.2.1.2 SSPS by Ant Colony Optimization

Xiao et al. [167] have used ant colony optimization (ACO) algorithm to assign tasks to humans. They run 30 random instances and show that their experimental results are promising and outperform previous genetic algorithm solutions. Chen and Zhang [33] used ACO algorithm to develop event-based scheduler model. They dealt with both human resource allocation and task scheduling where employees leaving/returning were regarded as events. In their work, it was known in advance when and how such events or variations would occur.

#### 2.2.1.3 SSPS by Other Evolutionary Algorithms

Minku et al. [111] proposed an improved evolutionary algorithm based on runtime analysis for the SPSP. They show that normalization is a key factor to get best results and how design choices in evolutionary algorithms affect project scheduling performance. Tavana et al. [153] dealt with three conflicting objectives simultaneously and introduced a new multi-objective multi-mode model. Xiuli et al. [165] proposed a hyper-heuristic algorithm for the SPSP. Luna et al. [110] and Chicano et al. [34] used a multi-objective evolutionary algorithm (MOEA) based on Pareto domination [40] to solve the SSPS problem by considering cost and duration as conflicting objectives. Valls et al. [154] proposed a hybrid GA for the resource-constrained project scheduling problem (RCPSP). In their work, the algorithm is improved rather than the project factors and scenarios. Hindi et al. [85] proposed an evolutionary algorithm to solve resource-constrained project scheduling problem. Their empirical study consists of 2,370 instances with 68% success (feasible solution obtained) rate, and with an average overall error rate of 0.95%. Penta et al. [55] presented a review of search-based optimization methods' applications for software project scheduling and staffing problem. Ferrucci et al. [60] multi-objective decision model helps software teams to balance the project risks and duration against overtime. Ren et al. [127] proposed a cooperative co-evolution approach which provides an optimal schedule for both team staffing and work package to achieve early overall completion time. Andrade et al. [48] proposed a hyper-heuristic based on SMPSO [117] configuration sets and grammatical

Table 2.1: Significance of Dynamic Events in Literature

| Dynamic Events | Significance |
|---|---|
| **Task-related**<br><br>1. Task addition<br>2. Task deletion<br>3. Task effort uncertainty | - Agile $2^{nd}$ principle [124] is:<br>"Welcome changing requirements, even late in development".<br>- Reasons for Late Software Delivery by Pressman [124].<br>"Changing customer requirements that are not reflected in schedule changes."<br>- Requirements uncertainty/velocity [107, 116]<br>- Underestimation of the effort amount and/or the required number of resources to do the job, by Pressman [124]. |
| **Employee-related**<br><br>4. Employee addition<br>5. Employee resigns<br>6. Employee leave<br>7. Employee return | - Staff turnover by Boehm [19]<br>- Human difficulties that seem not have been predicted, by Pressman [124] |

evolution to solve the SPS problem. In their study, no dynamic scenario was handled and results also lack evaluation on real-world data set. Besides these, some studies [128, 156] provide a survey of research on the SPS problem.

The above mentioned works present scheduling problems and optimal solutions under software project circumstances. However, none of these consider rescheduling problems during disruptions and schedule execution.

### 2.2.2 Dynamic Software Project Scheduling (DSPS)

All large-scale software projects are vulnerable to overruns and have risk and uncertainty factors. Software projects go through various unforeseen disruptions and dynamic events. The DSPS is an emerging area in the field of software engineering and has become a main focus of researchers' attention.

Table 2.1 explains the significance of dynamic events which may occur in real-world software projects. In this regard, Hepke et al. [77] presented a fuzzy model for the SPS system where activity time parameters were modelled by means of L-R fuzzy numbers due to uncertainty, and the fuzzy problem was transformed into a set of associate deterministic problems. A simulation based method aimed to limit the impact

of uncertainties on the overall project success, was proposed by Lazarova-Molnar and Mizouni [100] that selects the most appropriate remedial action scenario based on the project goal. Gueorguiev et al. [73] proposed a MOEA to provide the software project robustness under uncertainty. Robustness can be defined as the difference between the total time and estimated time when tasks durations are inflated or new tasks are added. The Pareto front found by their proposed MOEA represents the trade-off between project completion time and robustness. Antoniol et al. [6] used a tandem GA to process work packages in best sequence and allocate the staff to project teams in a best manner. Results obtained by GA were analysed by a queuing simulator. It guided the search process to determine that whether a negotiation of further people and successive iteration of the tandem GA process was required. To obtain an optimal solution, the whole process might repeat multiple times but their work does not consider stability and other dynamic events.

Ge [65] proposed a rescheduling method under uncertainty that deals with two objectives (efficiency and stability) and is based on GAs, but both objectives were handled as a single objective function by weighted sums. Xiao et al. [169] work deals with resource management under disruption prone environment such as requirement changes and urgent bug fixing. In their work, they used Little-JIL process definition language [163] to define software project activities in software development life cycle, activities dependencies, and need for resources. But there exist some limitations for their work. Firstly, they did not consider continuous rescheduling. Secondly, their work only incorporates capability constraints, availability constraints, and activity execution order constraints. Other constraints, such as different activities demanding the same resource are not considered. Their work also lacks some important objective functions as part of the scheduling problem.

Chicano et al. [34] work proposed a new multi-objective formulation for project scheduling and considered productivity of the employees at performing different tasks. They also provide robust solutions under analysis of the inaccuracies in task-cost estimations. Chand et al. [26] addressed resource-constrained project scheduling problem (RCPSP) with stochastic activity durations and resource availability under uncertain-

ties as a single-objective problem. Shen et al. [139] proposed a mathematical model with four objectives and three dynamic events (employee leaving/returning and new task addition) using a proactive-scheduling method (which has been explained in detail in next section). As an improvement of their work [137], they proposed a memetic algorithm dealing with five objectives with aforementioned dynamic events. There is still room to add more objectives and dynamic features in their work such as task precedence, task cancellation, and addition of new employee etc. Ge and Xu [67] proposed a GA and HC based optimizer. In their software project staffing model, they considered some dynamic elements of staff productivity. Furthermore, they considered both stability and efficiency as a single objective function by weighted sum. Their model also does not incorporate the interactions between parallel tasks.

The above literature highlights the fact that static approaches lack consideration of unpredictable scenarios while dynamic ones are also restricted to employee leaving and returning as events. In the real-world, many dynamic events could occur which make the scheduling problem highly complex. To resolve these issues, a more practical dynamic version of problem should be considered.

#### State-of-the-art Approach

Shen et al. [139] proposed a dynamic version of the SPS problem. In their model, they formulate the SPS problem by considering dynamic events and uncertainties that often occur during software project development. In this regard, they construct a mathematical model for the dynamic project scheduling problem with multiple objectives. They reason that the changing environments of software organizations mean that software project scheduling is a dynamic optimization problem. This model considers one uncertainty and three dynamic events. It deals with tasks efforts uncertainty which implies that modifications in task specifications by the customer or inaccurate initial estimates may cause changes in the initially estimated task effort. Moreover, a customer may raise new requirements requests during the software development life cycle. Regarding employees, the model also considers that employees can leave or return (from holiday) during the project execution. In addition to the constraints in the

model proposed by Alba and Chicano [5], this model considers that there should be a maximum number of employees assigned to a task to avoid communication overhead; however, it can be violated with a corresponding penalty if task skills are not fulfilled by allocated number of employees.

To handle the uncertainty and dynamic events, Shen et al. [139] also proposed a multi-objective approach. This method is based on $\epsilon$-MOEA [53] algorithm (explained in Section 2.1.2.2). A scenario-based approach is used to handle the task efforts uncertainty and very basic heuristic strategies are designed to handle the dynamic events.

### 2.2.3 Classification of Optimization Techniques used

In this section, we classify the above studies based on the optimization techniques. Out of these, 14(32%) studies [5, 29, 30, 170, 65, 75, 135, 66, 95, 172, 58, 149, 136, 169] used genetic algorithms as optimization technique. Ant colony optimization was used by 7(17%) studies [33, 167, 152, 74, 175, 46, 45] to solve the SPS problem. Particle swarm optimization technique was used by only 3(7%) studies [72, 90, 71]. Six (12%) studies [111, 139, 137, 127, 129, 76] used their proposed algorithm to solve this NP-hard problem. Hybrid algorithms were used by 6(15%) studies [73, 6, 20, 150, 168, 9]. Among these, 7(17%) studies [110, 34, 109, 35, 3, 108, 44] did comparison between different metaheuristic algorithms. Hence, we can conclude that genetic algorithm has been used widely to solve the software project scheduling problem. After genetic algorithm, researchers have used ACO to deal with this problem. Following pie chart (Figure 2.14) represents percentage of each algorithm used.

### 2.2.4 Data Sets used

In this section, data sets used by researchers to evaluate their proposed techniques are presented (Figure 2.15). In related work explained above, only 7(15%) studies have used real-world data [139, 137, 127, 73, 6, 95, 20] and 6(15%) studies [170, 65, 66, 71, 175, 9] considered case studies to evaluate their approach. Among these, 11(27%) studies [5, 111, 110, 167, 46, 168, 109, 35, 3, 108, 44] used Alba and Chicano instances [5]. Ten (22%) studies [29, 75, 149, 72, 58, 136, 74, 45, 76, 169] proved their approach by

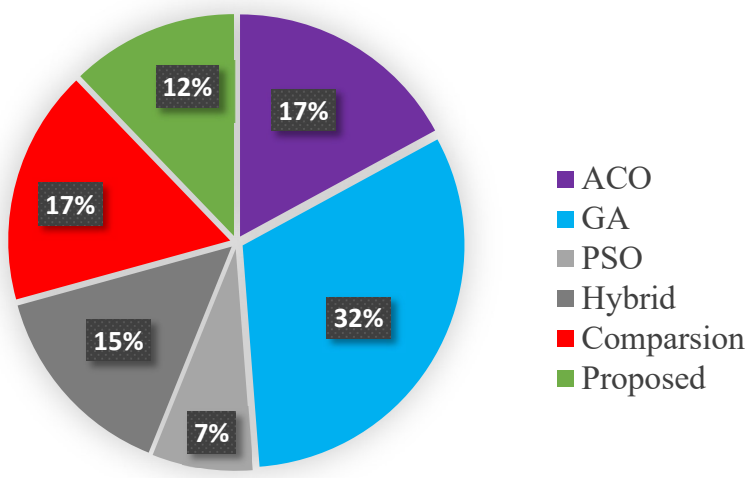



Figure 2.14: Existing Optimization Techniques for the SPS Problem

using examples. One study used combination of real-world and hypothetical data [150]. Two studies [139, 137] used real-world and benchmark data (Alba and Chicano [5]). Combination of real-world and randomly generated instances was used by 1 study [33]. Two studies [30, 170] evaluated their approach by using hypothetical projects. One study [90] used data from PSPLIB, and only one work [129] used data generated from simulation tool. Data generated by ProGen was used by 1 study [172]. Other real instances were considered by 1 study [34]. It can be seen that most researchers (27% studies) used Alba and Chicano instances [5] to evaluate their proposed approach and only few studies used real-world data. It is also concluded that real-world data should be used to evaluate an approach.

In this thesis, we have used benchmarked [139] and industrial datasets. The objective of benchmarked dataset is to validate the proposed model and approaches whereas industrial dataset provides evidence for its applicability in the real world situations.

Review of related work has given insight into certain problems. The SPS problem needs further research both on data level and algorithm level. The following major problems in current studies are identified:

i. Most of the studies deal with two objectives to be optimized usually time and cost.

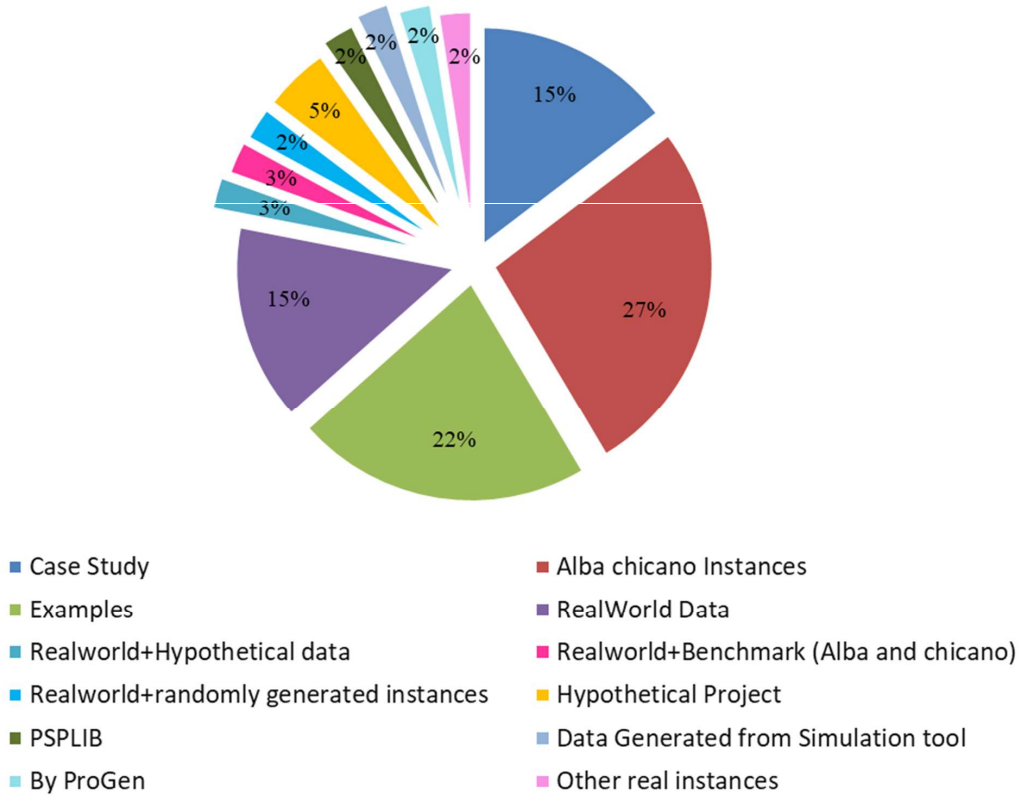



Figure 2.15: Data Sets used for the SPS Problem

ii. Lack of real-world data set usage while evaluating their techniques. Most of the studies have used case studies or some benchmark problems.

iii. Most of the studies deal with static SPS considering that no disruption occurs. Real-world and dynamic scenarios inclusion is ignored.

iv. In most of the studies, tasks' efforts are known in advance.

v. While proposing their technique, most of the studies do not give comparisons with other already existing techniques.

vi. Several studies have used the same problem formulation as defined by Alba and Chicano [5].

Next we describe the selected limitations to be addressed in this thesis:

i. **The lack of a conceptual model and approach to deal with a real-world dynamic event - Employee Turnover:** A software development project is associated with multiple uncertain events. One of them most frequently occurred is 'employee turnover' in software organizations. It can lead to failure of software projects if not handled well. Within this context, we advocate the development of a conceptual model and evolutionary approach to tackle this event (please see Chapter 3).

ii. **SPS overlooking a potential dynamic event - New Employee Addition**: Another drawback with existing studies is that they do not have capability to handle 'new employee addition' dynamic events which is one of the commonly occurred dynamic events in real-world situation. To fulfil this gap in current literature, we propose a new multi-objective evolutionary approach to deal with this dynamic event in an efficient and timely manner (please see Chapter 4).

iii. **Inclusion of important human attribute into SPS Model:** Human resources are the main resources in the SPS model. Current SPS models do not consider any human factor in their formulation, which may have significant impact on project duration and cost. Many studies have used the same model proposed by Alba and Chicano [5]. To overcome this limitation, we propose a model that incorporates an important human attribute 'employee experience' and formulates the cost objective according to real-world scenario (please see Chapter 5).

iv. **Real-world Data Set:** In this thesis, we use real-world data with maximum 91 tasks to solve the SPS problem. Previous studies address the SPS problem with 15 tasks for real-world data.

## 2.3 Summary

This chapter presented main concepts, features and applications of software project scheduling problem and search-based software engineering with an objective to provide background and to situate the problem addressed in this research thesis. First, an

overview of the problem structure was presented. Then, the related literature was reviewed, focusing on both static and dynamic studies. This analysis helped to reveal an existing trend to deal with the SPS problem and justified the opportunity to build a general model and approaches that could be easily adapted to solve this problem under a dynamic environment. These approaches should be flexible to accommodate different kinds of dynamic events.

In short, the above studies mainly use static approaches to deal with the SPS problem. They may face performance deterioration and infeasibility when faced with disruptions. In addition, few studies have led to dealing with some dynamic events (Employee leaving/returning, new task addition) and demonstrated promising results. However, still there is need for great improvements before the approaches proposed in these studies can be considered as an effective tool for the SPS problem under a dynamic environment. This suggests a significant need for new methodologies to deal with other real dynamic events in the area.

# Chapter 3

# A Systematic Approach to Deal with 'Employee Turnover' Dynamic Event

Today, there is extensive evidence that software organizations are experiencing a competitive market where managing cloud based software solutions (large scale software projects) is tedious and error-prone [139]. Hence, it necessitates the need for computer-aided tools for effective project planning within time and budget considering volatile project parameters. Employee turnover is one of those volatile project parameter which negatively affects the project schedule and budget. Therefore, it has become a significant topic for scholars, academics and project managers.

This chapter is dedicated to the handling of 'employee turnover' as a potential dynamic event that may occur during software project development and has not been considered before in the literature. The employee resignation can cause tangible and intangible loss for the software company, and significantly impact the project schedule and cost, if not addressed in time.

This chapter presents a model to deal with 'employee turnover' as dynamic event which is critical for projects success. It also presents a multi-objective evolutionary approach to handle this dynamic event. A relevant and common set of features are

considered for the software project scheduling (SPS) problem. The aim of our study is to regenerate an appropriate schedule with minimum project duration and cost objectives when an employee resigns from the company.

The remainder of this chapter is structured as follows. Section 3.1 presents the brief background for 'employee turnover'. Section 3.2 introduces the motivation behind 'employee turnover' as dynamic event. Section 3.3 presents the model and evolutionary multi-objective approach to deal with this dynamic event. In Section 3.4, problem objectives and constraints are defined. Section 3.5 is devoted to experimental studies. Finally, Section 3.6 concludes the chapter.

## 3.1 Background

The software industry has marked a rapid growth in the last two decades. However, in this growth a high ratio of project failure and overruns has been observed. The underlying reasons for these unsuccessful projects are noted as management issues. This has raised interest in the software project management activities responsible for project success and failure.

In real-world, various dynamic factors, such as employee turnover, skills of engineers, new employee addition, and task effort uncertainty could influence the development of software project. In this case, the effective resource allocation is crucial for software project managers when managing medium to large scale projects development, to meet the budget and schedule constraints [139]. This refers to a software project scheduling problem (SPSP) that answers which employee will perform which task during the software development life cycle [5]. In general, software project scheduling can be defined as appropriate allocation of employees to tasks on a time scale. It is important that while doing allocations task dependencies and constraints must be satisfied. Unlike projects in other fields, software projects comprise people-intensive activities where skilled human resources are the main resources [33]. Therefore, a suitable and correct assignment of employees to software tasks is essential for the success of software development organizations.

The SPS problems are composed of tasks, employees, constraints, objectives, and task dependencies; where a schedule's feasibility depends on the constraints and optimality is defined by project objectives. For a multi-objective problem, constraints must be satisfied whereas objectives should be satisfied. In this regard, a feasible project schedule satisfies all the constraints whereas an optimal project schedule satisfies both constraints and objectives. The optimal schedule is also much appropriate among all feasible schedules with respect to objectives.

In the recent time, there have been concerns on the reasons and consequences of employee turnover within the software industry. The notion of turnover culture has emerged and can enhance our understanding about turnover in software industry. An employee turnover is defined as the workers' rotation around the labour market; between firms, jobs and occupations; and between the states of employment and unemployment [1]. Price [125] defines the term 'turnover' as: the ratio between number of people who have resigned to the average number of organizational members in that organization during the specified period. According to statistics, the employee turnover rate in some organizations has increased from the traditional 6∼10% to 25% [114], causing real alarm. The reasons for high turnover rates of IT professionals depend on both internal organizational factors (for example, poor jobs, inadequate career paths) as well as market demand factors (for example, lucrative salaries, strong market demand for software professionals) [2]. Therefore, the average time of developer incumbency is approximately 13 months [114], which shows that employees are less loyal to their employers and more loyal to their career and skills [56]. Besides the fact that resignations present big challenges for project managers, the resignation of a core employee who is deeply integrated into the project can delay the project or even prevent the implementation of new technology or system. Ultimately, the project may be stopped and business opportunity may be lost. According to [4], the following factors may trigger the employee turnover in any organization:

- Managerial factors. Employees are more inclined to stay and work in an organization which is stable and where the working environment is friendly.

- Working environment. Lack of basic facilities may result in a low-grade working environment, consequently causing employee turnover.
- Pay. Lower salary is considered as one of the critical factors for employee turnover.
- Fringe benefit. Lack of fringe benefits and employees' perks are also reasons for employee turnover.
- Career promotion. The best motivational factor for employees is career promotion which includes higher salaries etc.
- Job fit. Another major cause of employee's turnover is his/her dissatisfaction with the job. Organizations productivity is increased if more suitable employees are recruited and necessary measures are applied to increase job satisfaction.
- Clear job expectation. Turnover can be minimized by meeting job expectations for the newly hired employees.
- Perceived alternative employment opportunity. One of the unmanageable issues for an organization is that when employees leave the organization if there is a possibility to get another job.
- Influence of co-workers. A turnover culture in an organization will also trigger more and more employee turnover decisions.

## 3.2 Motivation

Turnover as a result of resignations of employees raises costs, reduces production, disrupts teams, and results in lost knowledge. Therefore, managing turnover successfully is a must to achieve the project success. In the literature, very few studies [33, 139, 137] address multi-objective software project scheduling under a dynamic context. Especially, no work considers 'employee turnover' as dynamic event. Although the literature on this topic is vast but most of the researchers have focused on the causes of employee turnover, and about predicting and measuring the probability of employee turnover

rate [130]. This motivates our research in developing a new model and approach to tackle this dynamic event.

With the aim of completing a project within budget, a software project manager may preferably reschedule existing employees to avoid any training cost and hiring delay provided that existing employees have the competency to fulfil project needs. It is also essential that the project manager is happy with reschedule solutions. Otherwise, company recruits a new employee. It is not always easy for the manager to hire an employee with exactly the same skills as resigning employee. As resources with required skill set may not be readily available in the market; hence, a manager may need some flexibility in employee's skills while considering time and budget constraints. Therefore, it is critically important to handle this dynamic event.

In our study, we present a model when an employee turnover occurs. This model considers both employee rescheduling and recruitment scenarios. We also present a multi-objective evolutionary algorithm (MOEA) to deal with this practical dynamic event for the SPS problem. The proposed approach applies the concept of $\epsilon$-MOEA [53] and uses domain knowledge. It optimizes and reduces project duration and cost objectives. In case of employees' rescheduling after the evolutionary search process and optimization, we provide a set of solutions to project manager for decision making. It is up to the manager that he/she is satisfied with such solutions. The proposed approach also helps the project manager in hiring a new employee if remaining employees after employee turnover could not fulfil project skills or the manager is not happy with rescheduling results. This part of approach also incorporates manager's provided information in the search process for the software project scheduling problem. Once the search process is completed candidate solutions are given to the decision maker to select the final schedule indicating the minimum proficiency level for new employee. It could be a real challenge for software project manager to find a quality software professional without any hiring delay. This hiring delay may ultimately affect project duration and cost. The purpose of this research, therefore, is to propose an approach to address these identified issues.

## 3.3 The Proposed Approach

### 3.3.1 Employee Turnover Model

According to Boehm [19], employee turnover is an important reason for the failing on software projects. It can undermine the organization's productivity, success and stability. Therefore, it is more significant to conduct the research on this topic to help business organizations by identifying their problems, analysing the information and recommending possible solutions. Employee resignation can cause tangible and intangible loss for the software company, and significantly impact the project schedule and cost if not handled well. Considering these facts, we present a model (Figure 3.1) based on real-world scenario when an employee decides to resign from the software organization during project execution.

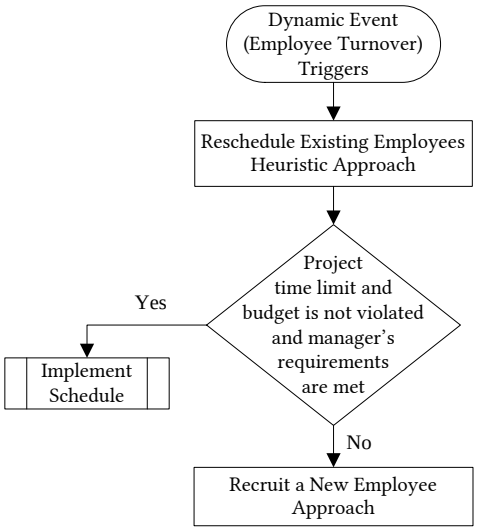



Figure 3.1: Employee Turnover Model

This model has two parts: i) employees rescheduling, ii) new employee recruitment. First, the model tries to reallocate existing employees on effected tasks if they could cover required skills without violating the project's time limit and budget. If the existing employees are unable to fulfil the project needs and employees end up overworking, then the model provides the manager with a set of options about what kind of person

he/she could recruit within the budget. These are elaborated in detail below.

### i. Rescheduling Approach

In the rescheduling approach, the existing employees are rescheduled. For this purpose, a heuristic strategy is designed as close to the real-world scenario. According to the heuristic, if the resignation of an employee affects some tasks, then reallocate an employee who could fulfil the missing skills required by that task given that employee overwork constraint is also satisfied. Furthermore, if two employees could be allocated to an effected task, then two different heuristic solutions will be generated. This heuristic strategy provides a decision support to the software project manager. The pseudo code for reschedule heuristic is given in Algorithm 1.

One possible reason for rescheduling approach is that the resigning employee may be the least proficient for the tasks that does not cause too much delay in the project.

**Algorithm 1:** Reschedule Heuristic

**Input :** $E\_rem\_set$: remaining employees, $T$: set of available tasks, $X$: dedication matrix

**Output:** $R_{hs}$: Heuristic solutions with rescheduling of existing employees

```
for ∀T_j ∈ T do
    T_eff_set = Effected tasks /* a subroutine findEffectedTasks() is called to
     find all the tasks which have been affected by employee resign */
end
for ∀t_j ∈ T_eff_set do
    for ∀e_i ∈ E_rem_set do
        if e_ij^prof > 0 then
            X_ij : assign employee e_i to t_j with maximum dedication.
        end
        if Task_skill_constraint(X_ij) = 0 & Overwork_constraint(X_ij) = 0
         then
            X_ij : assign employee e_i to t_j with maximum dedication. /* a
             heuristic solution is generated */
        else
            exit; /* Process is terminated and new employee hiring approach is
             adopted */
        end
    end
end
return R_hs
```

Another reason is that hiring a new employee may cost additionally. According to one estimate, the time for new software employee to become maximally effective is 18 months [99]. A large stream of new people into a project will incur the higher training overhead [2] and communication overhead. It is known fact that communication overhead is directly proportional to $n^2$, where $n$ is the size of the team [134, 140].

Evolutionary algorithms (EAs) provide a non-dominated solutions set as a result to solve any NP-hard problem [35]. Therefore, after rescheduling, we provide a set of feasible solutions to the software project manager. The manager decides whether he/she is happy with such solutions. It will enable him/her to make informed decisions. A software project manager may look at three types of solutions, namely, best duration, best cost, and best trade-off among all objectives.

### ii. New Employee Hiring Approach

If the resigning employee has some specific skills, which none of the existing employees possess or project manager is not satisfied with rescheduling solutions; then the company recruits a new employee having those skills. Finding quality software professionals without delaying the project can be a real challenge for a project manager.

To address these issues, in this approach (Figure 3.2), we can provide the software project manager with information, what kind of person they can hire. This method will help the manager in recruiting a new employee without any hiring delay. It is assumed that a new employee is hired somewhere in the middle of the project; however, Brook's law does not apply which states that adding manpower to a late project makes it later [21]. It is also assumed that the employee's minimum proficiency value related to a skill is '0.1' (see Chapter 2.1.1 for details). The steps for this procedure are given below:

**Step 1**: The software project manager's provided information (threshold values: project duration and cost values) is taken as input.

**Step 2**: Extract knowledge from the skills of the resigning employee and feed into the system.

**Step 3**: Vary respective skills proficiencies for optimization. This variation is started with the lowest proficiency value that an employee could have. Depending

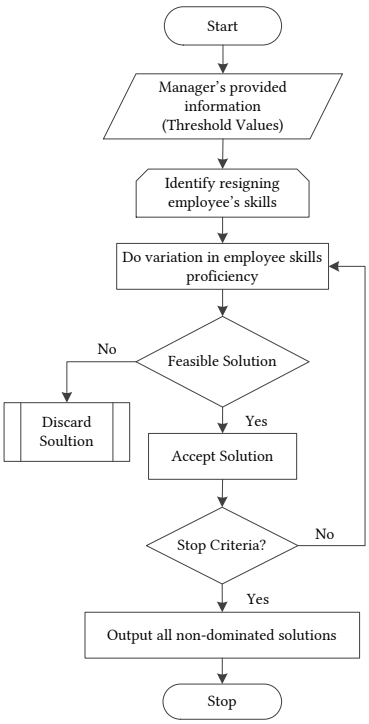
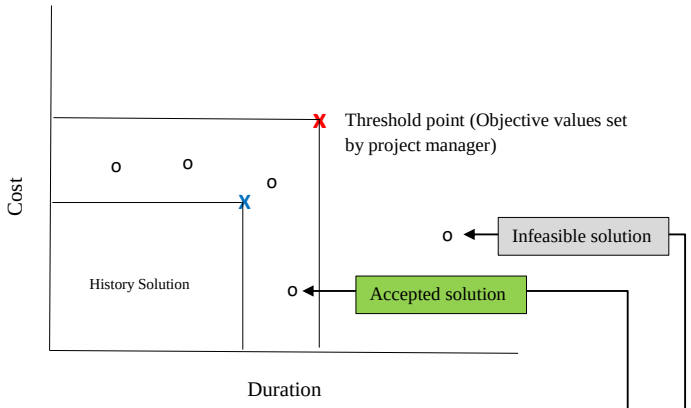



Figure 3.2: Working Mechanism to Hire a New Employee

on the number of critical skills which have significant impact on project, first lower one skill's proficiency value and see the result. If there is success, then, lower the second skill's proficiency and so on. Based on all previous results, we increase/decrease proficiency value step-wise. As a result, we will have a representative set of all possible solutions. Let $a$ and $b$ are two critical skills which have significant impact on project. Then, we first lower skill $a$ proficiency value and based on the outcome, we lower second skill's proficiency value as shown in Figure 3.3.

**Step 4**: After the search process and optimization, if the result is within the threshold region, then the solution is accepted. In other case, if the solution is out of threshold limit then it is discarded. Initially, the threshold value is same as previous duration and cost values. If no solution is found with lowest possible proficiency values, then, we increase threshold value and see result. The software project manager may not want to exceed the duration and cost values beyond a certain limit. This threshold value will be set by the project manager himself/herself.




**Proficiency Variation Matrix**

| Skills | a | b | c | d |
|---|---|---|---|---|
| Proficiency | 3.5 | 4.0 | 4.9 | 5.0 |
| | 0.1 | 4.0 | 4.9 | 5.0 |
| | 0.1 | 0.1 | 4.9 | 5.0 |

Figure 3.3: Working Mechanism of Proficiency Variation

### 3.3.2 MOEA-based Method for SPS Problem

This section presents an evolutionary multi-objective approach to handle the 'employee turnover' dynamic event. The mechanism of proposed approach is based on $\epsilon$-MOEA [53]. The proposed MOEA method handles: i) rescheduling of employees, ii) new employee hiring.

For rescheduling, it uses domain knowledge to generate the initial population. Creation of an initial population in population-based algorithms is an important and major step to make the generation more progressive. It is a crucial task in evolutionary algorithms (EAs) as the convergence speed and the final solution's quality can be affected. Many EAs do not use domain knowledge to create the initial population, and random solutions is the most commonly used method for this purpose. Domain knowledge avoids searching unnecessary regions, producing more promising results, and boosting EAs convergence speed. Therefore, in our proposed approach (Figure 3.4), at scheduling point $t'$ when dynamic event occurs, the initial population is generated using the following combination of solutions:

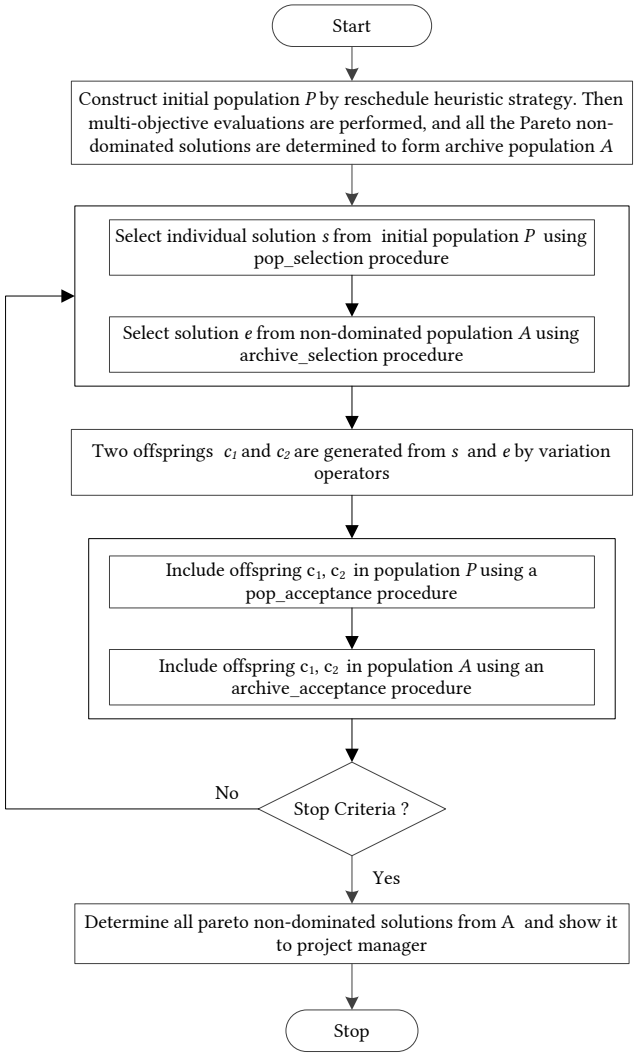



Figure 3.4: Procedure of Rescheduling algorithm at scheduling point '$t$'

**Case a.** If leaving employee affects some tasks then respective heuristic solutions are generated (see Section 3.3.1 for details).

$$X\%Heuristic + Y\%Baseline + Z\%Random \tag{3.1}$$

In our case, 50% are heuristic solutions and its variants using mutation operator. 1% is pre-schedule/baseline/history solution (implemented schedule) and 49% are random solutions. The inclusion of baseline solution will guide the search process to be closer to the implemented solution. We are using a trade-off between random initialization

and heuristic procedure. Random initialization will give diversification on the solutions while heuristics/baseline solution will converge the solutions rapidly.

**Case b.** If no heuristic solution is generated. It means that remaining employees could fulfil skills for each task.

$$X\%History + Y\%HistoryVariants + Z\%Random \tag{3.2}$$

In Eq. (3.2), 1% is pre-schedule/baseline/history solution (implemented schedule), 50% are its variants using mutation operator and 49% are random solutions.

For new employee hiring, the proposed MOEA incorporates manager provided information during the search process. We also set up the information for new employee (variation in employee skill's proficiency values and this variation is started with lowest value that an employee could have). After the search process and optimization, candidate solutions (who meet the threshold values with shuffling of one to two employees) are provided to decision maker. Decision maker selects the best trade-off among them by providing the minimum proficiency level for new employee. A general flow chart of this procedure is explained in Figure 3.5. For 'new employee hiring' approach (Figure 3.6), initial population is formed using the following combination of solutions:

$$X\%History + Y\%HistoryVariants + Z\%Random \tag{3.3}$$

In Eq. (3.3), 1% is history solution with the information set up for new employee, 50% are its variants using mutation operator and 49% are random solutions.

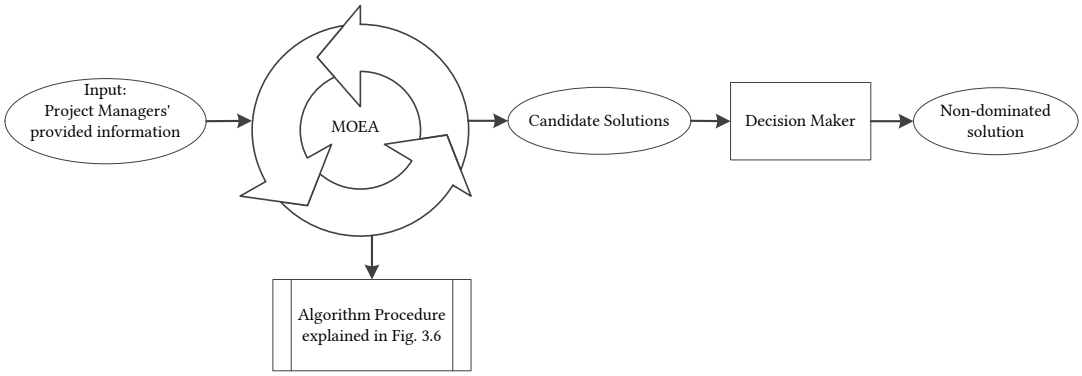



Figure 3.5: General Mechanism of 'New Employee Hiring' MOEA

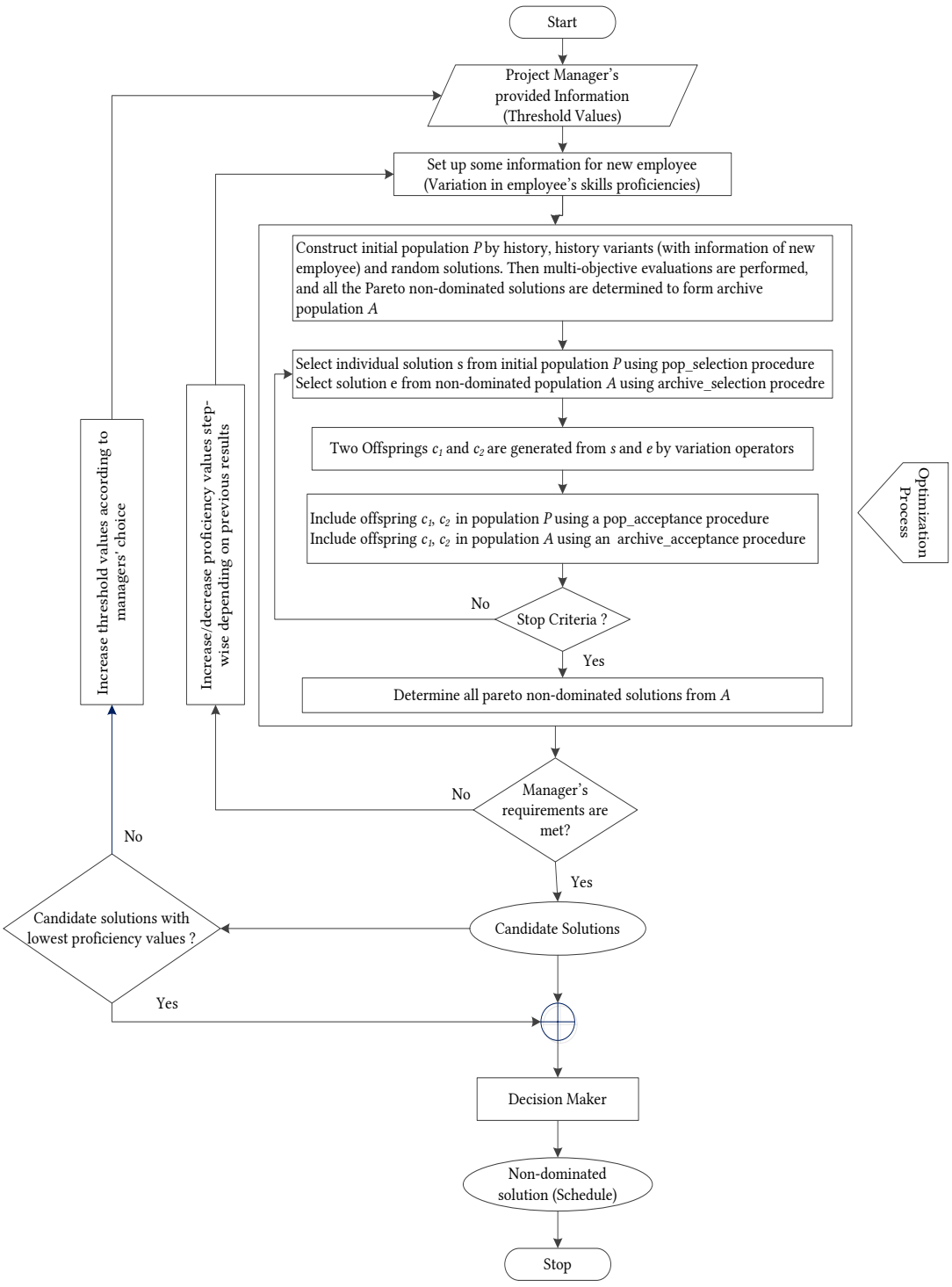



Figure 3.6: Procedure of 'New Employee Hiring' algorithm at scheduling point '$t$''

## 3.4 Optimized Objectives and Constraints

### 3.4.1 Objective Functions

This section formulates the project objectives namely, duration and cost. The objectives are optimized to see how our proposed approach works for a new dynamic event. The mathematical formulation and detail of each objective is according to [139].

$$\min \mathbf{F(t)} = [f_1, f_2] \tag{3.4}$$

The project duration and cost objectives are denoted as $f_1$, $f_2$ respectively.

$$f_1(t) = duration = max(T_j^{end}) - min(T_j^{start}) \tag{3.5}$$

The duration measure $f_1(t)$ in Eq. (3.5) is maximum elapsed time required for completion of each available task. $T_j^{end}$ and $T_j^{start}$ denotes the start time and end time of each task respectively. Moreover, duration for the whole project is the maximum finishing time of the last task.

$$f_2(t) = cost = \sum_{e_i \in E\_ava\_set} e\_cost_i \tag{3.6}$$

$f_2(t)$ in Eq. (3.6) represents project cost, which is sum of all expenses paid to the available employees against their dedications for the available tasks. The set of available employees is denoted by $E\_ava\_set$. Let $T\_active\_set$ denotes the set of actives tasks which are being developed at time moment $t'$, where $t'$ represents any month during which the project is being developed. A task is called active if it does not have any preceding unfinished task in the TPG at time $t'$. Hence, the expenses paid to the employee $e_i$ at $t'$ month are calculated as follow:

$if \sum_{j \in T\_active\_set} X_{ij} \leq 1.0$

$$e\_cost_i = e_i^{norm\_salary} \cdot t' \cdot \sum_{j \in T\_active\_set} X_{ij} \tag{3.7}$$

$if \sum_{j \in T\_active\_set} X_{ij} \geq 1.0$

$$e\_cost_i = e_i^{norm\_salary} \cdot t' \cdot 1 + e_i^{overwork\_salary} \cdot t' \cdot \left( \sum_{j \in T\_active\_set} X_{ij} - 1 \right) \tag{3.8}$$

where $e_i^{norm\_salary}$ is an employee's normal salary, $e_i^{overwork\_salary}$ represents overwork salary, $X_{ij}$ is dedication of an employee $e_i$ to task $t_j$. $X_{ij}$ greater than 1 indicates that employee overworks for the project and overtime salary is also added to total salary.

### 3.4.2 Constraints

The SPS problem is subject to following constraints [139]. Below (i)-(iii) are hard constraints while (iv) is a soft constraint.

i. All tasks allocation constraint. No task should be left unassigned. Each task should be allocated to at least one available employee.

$$\forall e_i \in \text{e \_available \_set(t)}, \sum T_j\text{\_Dedication} \neq 0 \tag{3.9}$$

ii. Task skills constraint. All the available employees allocated on a task must collectively cover all the task required skills.

$$T_j\text{\_Skills} \subseteq \cup e_i \ \{ \ \text{Skills} \ | X_{ij} > 0 \ \} \tag{3.10}$$

iii. No overwork constraint. No employee overworks for a project means that employee should not work for the project more than his/her maximum dedication.

$$\sum_{j \in T} X_{ij} \leq e_i^{max\_ded} \tag{3.11}$$

iv. Task head count constraint. According to the best practices of software engineering development, the number of employees assigned to a task should not exceed a certain limit. Task headcount is calculated according to [19].

$$\forall T_j, T_j^{no\_of\_emp} \leq T_j^{max\_headcount} \tag{3.12}$$

### 3.4.3 Employee's Proficiency Calculation

The proficiency of an employee $e_i$ for a task $t_j$ can be defined as the employee's capability level to tackle that task. It is denoted as $e_{ij}^{prof}$ and ranges between [0,1]. According to Chang [30] calculation, if an employee has zero proficiency against a skill related to a task then that employee's total proficiency for performing that task becomes zero, which should not be a case. In reality, if an employee's proficiency for one skill is zero while having other skills for a task, then the employee could still perform that task. Therefore, in this work, we define employee to task proficiency as:

$$e_{ij}^{prof} = \sum_{k \in req_j} \frac{prof_i^k}{C} \tag{3.13}$$

In Eq. (3.13), $req_j$ denotes task required skills and each skill has a proficiency value against it; $C$ denotes a constant and we set '$C = 5$' according to [30]. We then calculate average proficiency as follows:

$$e_{ij}^{prof} = \frac{e_{ij}^{prof}}{total\_skills\_required\_by\_task} \tag{3.14}$$

## 3.5 Experimental Results

This section answers the following research questions by empirically investigating the proposed method:

i. RQ1. Can existing methods deal efficiently with the dynamic events 'employee turnover'?

ii. RQ2. Does rescheduling of existing employees delay the project upon employee turnover?

iii. RQ3. What kind of employee is hired if duration and cost are extended? If not, then what should be the level of new employee skill proficiency?

### 3.5.1 Experimental Setup

#### DSPS Instances

In this study, we use 18 dynamic benchmark and 6 real-world data instances.

***Benchmark.*** These 18 dynamic instances are derived from Alba and Chicano's benchmark [5] which are gathered from different software projects. The reason for choosing this benchmark data set is that these instances include variants of three important factors (number of employees, number of tasks and number of employee skills) for the SPS problem as in real-world scenarios. These dynamic data instances differentiate themselves from the static ones in [5] with the following keys aspects: task maximum headcount, task effort uncertainties, part-time jobs, and overworking of employees. In the project, it is assumed that part-time employees are 20 percent of the total employees whose maximum dedications are in the interval [0.5, 1); another 20 percent of employees do overtime work, whose maximum dedications are generated uniformly from (1, 1.5] at random; and remaining employees are full time, their maximum dedication is set to 1.0. Each employee has an associated proficiency score for a skill; these scores are sampled uniformly from (0, 5]. Following the practice in [5], each employee's normal monthly salary is sampled from a normal distribution with the mean of 10,000 and standard deviation of 1,000. When the project starts, a task precedence graph (TPG) is generated using the method in [5].

***Real-world.*** Six real-world instances are also used in our experiments. Three of these instances are derived from business software construction projects for a departmental store [70]. We also use 3 real instances obtained from a software company. These data instances comprise of 10,8,5 employees and 42,43, and 91 tasks respectively.

The data instances are denoted as 'T10_E5_SK4-5', where 'T10' means total number of tasks in the project, notation 'E5' represents total number of employees and 'SK4-5' means number of skills of an employee. The six real instances are named as 'Real_n_ Tx_Ey' where n $\in$ {1,2,...6} and x, y represents number of tasks and number of employees respectively. To give an illustration of real-world data instances, we present employees and tasks properties in Table 3.1 and Table 3.2 respectively.

Table 3.1: Employees Properties for Real_3 data instance

| ID | Is Contract | Basic Salary | Overwork Salary | perHour Salary | nHours | Exp | Ded |
|---|---|---|---|---|---|---|---|
| 1 | N | 3400 | 105 | 35 | 100 | 0.82 | 1.25 |
| 2 | N | 2400 | 102 | 34 | 100 | 0.69 | 1.5 |
| 3 | Y | 0 | 228 | 114 | 100 | 0.59 | 1.75 |
| 4 | N | 3200 | 129 | 43 | 100 | 0.8 | 1.5 |
| 5 | N | 2500 | 0 | 27 | 100 | 0.48 | 0.5 |
| 6 | Y | 0 | 0 | 50 | 100 | 0.63 | 0.5 |
| 7 | N | 4100 | 0 | 43 | 100 | 0.94 | 0.5 |
| 8 | Y | 0 | 0 | 79 | 100 | 0.32 | 0.25 |
| 9 | Y | 0 | 0 | 109 | 100 | 0.91 | 1 |
| 10 | N | 3500 | 0 | 38 | 100 | 0.9 | 0.25 |

Table 3.2: Tasks Properties of Real_3 data instance

| Task ID | Status | Effort | HeadCount | Skills |
|---|---|---|---|---|
| 1 | 1 | 3 | 6 | 1,3,5 |
| 2 | 1 | 2 | 3 | 2 |
| 3 | 1 | 2 | 5 | 4,5 |
| 4 | 1 | 2 | 4 | 1,4,5 |
| 5 | 1 | 3 | 6 | 3,5 |
| 6 | 1 | 1 | 3 | 4 |
| 7 | 1 | 4 | 6 | 4,5 |
| 8 | 1 | 3 | 3 | 1,2 |
| 9 | 1 | 2 | 6 | 1 |
| 10 | 1 | 2 | 5 | 1,4 |
| 11 | 1 | 2 | 3 | 2,5 |
| 12 | 1 | 3 | 3 | 3 |

**Parameter Settings**

For a rational comparison among algorithms, a set of parameter values are presented in Table 3.3.

In this chapter, for each algorithm on each problem instance (18 benchmark and 6 real-world), 30 independent runs were executed to obtain all the results.

### 3.5.2 Performance Analysis of Existing Algorithms

This section explores whether existing approaches can deal with 'employee turnover' dynamic events efficiently. We investigate both state-of-the-art (SOA) [139] (see Chapter 2.2.2 for details) and a baseline algorithm for new dynamic events. The baseline algorithm is a MOEA-based complete rescheduling method [119] and has been attributed

Table 3.3: Parametrization (L = Individual Length)

| **State-of-the-art algorithm [139]** | |
|---|---|
| Population size | 100 individuals |
| Crossover | SBX, pc = 0.9 |
| Mutation | polynomial, pm = 1.0 / L |
| No of evaluations | 10,000 |
| **Baseline Approach** | |
| Population size | 100 individuals |
| Crossover | SBX, pc = 0.9 |
| Mutation | polynomial, pm = 1.0 / L |
| No of evaluations | 10,000 |

to regenerate a new schedule from scratch. It indicates that, at each scheduling point, the initial population is randomly generated. We evaluate for a diverse set of data instances. The results are depicted by a clustered stacked graph in Figure 3.7. One column in the graph represents one objective for one data instance. The first objective is cost and$2^{nd}$ column represents duration. We normalize objective values to be on the same scale. It is obvious from results that there is no clear pattern for both algorithms to deal with new dynamic events. For some instances, the baseline outperforms the state-of-the-art algorithm and vice versa. All these key factors necessitate the need for a new approach to deal with 'employee turnover' dynamic event.

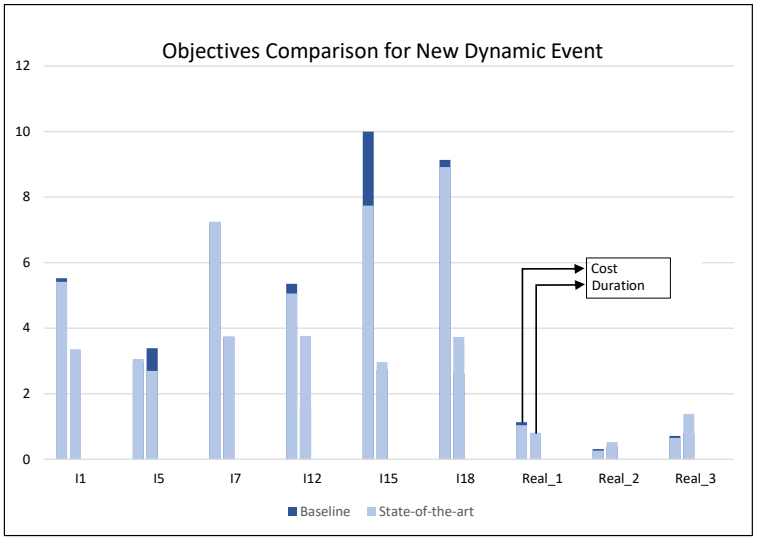



Figure 3.7: State-of-the-art and Baseline algorithm performance for 'Employee Turnover' dynamic event

Table 3.4: Duration and Cost Values Comparison for Real Data Instances. The selected solution against each instance is according to Decision Maker result [138].

| | Real_1 | | Real_2 | | Real_3 | |
|---|---|---|---|---|---|---|
| | SOA | Baseline | SOA | Baseline | SOA | Baseline |
| Duration | 8.12E+00 | **6.20E+00** | 5.22E+00 | **3.91E+00** | 13.9E+00 | **7.99E+00** |
| Cost | **1.52E+05** | 1.65E+05 | **3.81E+04** | 4.55E+04 | **9.55E+04** | 1.04E+05 |

We also present objective values for real data instances against each algorithm for 'employee turnover' dynamic event in Table 3.4. Results indicate that baseline outperforms state-of-the-art algorithm for the 'duration' objective for all the real instances. For the 'cost' objective, the state-of-the-art overcomes the baseline algorithm while giving minimum cost.

Evolutionary algorithms (EAs) provide a non-dominated solutions set as a result to solve any NP-hard problem [35]. Therefore, we only present a single solution from the solution set according to the decision maker (DM) choice [138] (see Chapter 2.1.2.3 for details). In practice, a non-dominated solutions set found by EAs is provided to software project manager subject to manager's choice. The project manager may select a solution considering best duration, best cost or best trade-off among all project objectives. However, the involvement of a person for taking decisions is not practical in our experiments. Therefore, an automated decision making method [138] is adopted.

### 3.5.3 Project Delay by Rescheduling Existing Employees

This section investigates whether rescheduling of existing employees using our proposed approach (after employee resignation) significantly delays the project, provided existing employees have the capability to fulfil the required project skills. At the scheduling point $t'$, heuristic solutions are generated for a task if the leaving employee affects it. These solutions are variants of existing employees' allocations who can perform that task by fulfilling the task's required skills. For example, if two employees could be allocated to an effected task, then two different solutions will be generated. In other case, if an employee resigns and all other allocated employees could fulfil task skills then no heuristic solution is generated. In this case, history solutions and its variants are used.

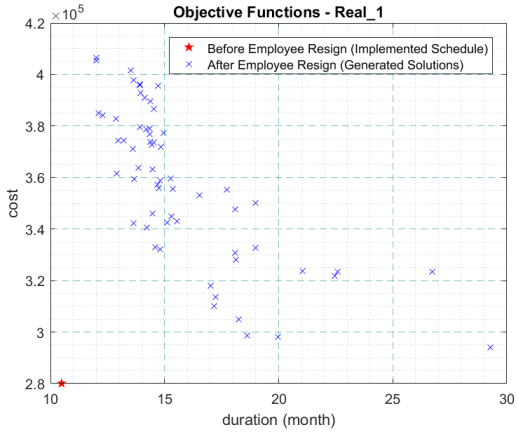
Table 3.5: Duration and Cost Values Comparison for Data Instances after Rescheduling. The selected solution is the average of 30 independent runs.

| | Before DE | After DE | Before DE | After DE | Before DE | After DE |
|---|---|---|---|---|---|---|
| | Real_1 | | Real_2 | | Real_3 | |
| Duration | 1.26E+01 | 1.98E+01 | 9.79E+00 | 1.00E+01 | 2.25E+01 | 3.12E+01 |
| Cost | 2.68E+05 | 2.91E+05 | 9.12E+04 | 8.97E+04 | 1.72E+05 | 1.85E+05 |
| | T10_E5_SK4-5 | | T20_E5_SK4-5 | | T20_E10_SK4-5 | |
| Duration | 4.93E+01 | 9.69E+01 | 5.10E+01 | 9.85E+01 | 4.74E+01 | 6.02E+01 |
| Cost | 1.01E+06 | 1.62E+06 | 1.52E+06 | 2.17E+06 | 1.71E+06 | 1.92E+06 |
| | T30_E10_SK4-5 | | T30_E15_SK4-5 | | T30_E15_SK6-7 | |
| Duration | 4.27E+01 | 5.73E+01 | 5.40E+01 | 6.18E+01 | 5.48E+01 | 6.83E+01 |
| Cost | 2.11E+06 | 2.27E+06 | 2.71E+06 | 2.86E+06 | 2.59E+06 | 2.56E+06 |

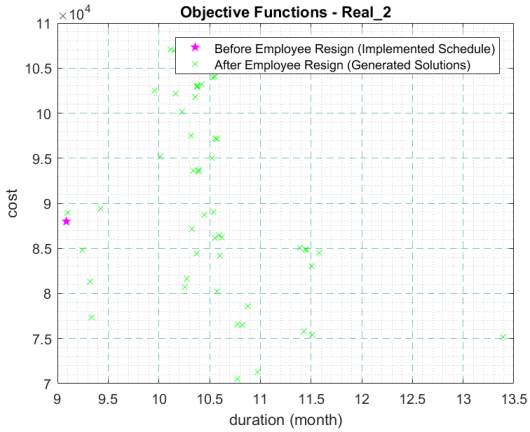
We present results (Table 3.5) for benchmark and real data instances for duration and cost objectives after rescheduling. The results indicate that after the 'employee turnover' dynamic event (DE) occurrence, for all the instances, duration is increased. Often, the duration of a project goes hand in hand with the cost. Real_1 and Real_3 have a significant increase in duration; therefore, cost is also increased. For Real_2, project delay is small and cost is reduced. For benchmark data instances, both duration and cost has been increased except instance 'T30_E15_SK6-7'. Although, for this instance the duration objective shows a significant increase but cost is reduced. The reason is that the leaving employee had a really high salary as compared to others; same applies for Real_2. These results are also visually represented for some instances in Figure 3.8. It is the manager choice that either he/she is happy with such delay in project.

### 3.5.4 Skill level for New Hired Employee

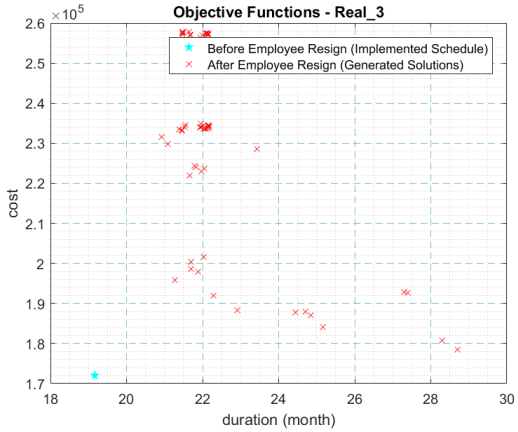
When an employee resigns, then, an employee with exactly the same skills (as resigning employee) may not be readily available in the market. We investigate using our proposed approach that what could be the lowest proficiency value of the new hiring employee if the software project manager extends both project duration and cost values. On the other hand, if manager decides to keep the threshold same as previous, then, what kind of employee could we hire? To explore this, we use three real instances namely, Real_4, Real_5, Real_6. If an employee gives resignation, Real_4 has one missing skill (SK: 9) which none of the existing employees possess; Real_5 and Real_6 has



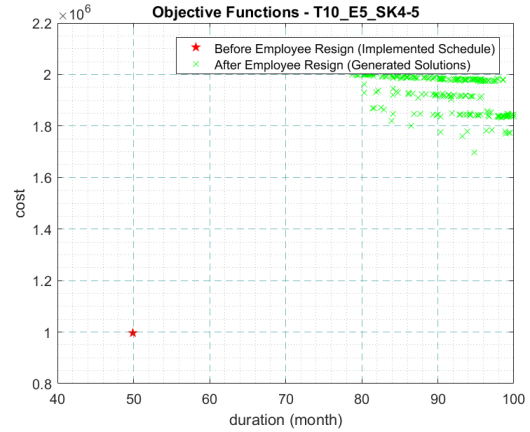

(a) Real_1



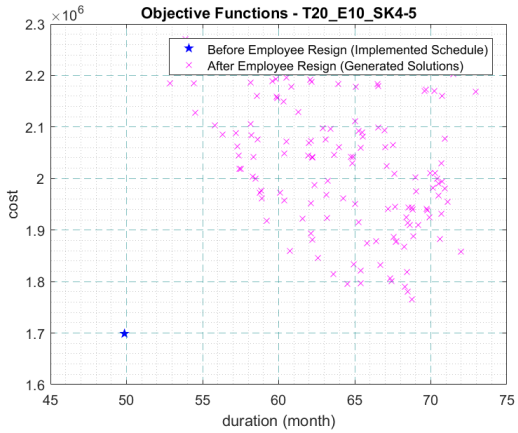

(b) Real_2



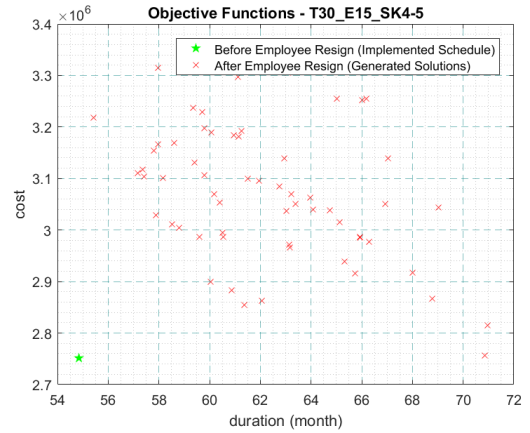

(c) Real_3




(d) T10_E5_SK4-5




(e) T20_E10_SK4-5




(f) T30_E15_SK4-5

Figure 3.8: Rescheduling of Existing Employees for Data Instances.

Table 3.6: Skill's Proficiency Values for New Hiring Employee

| Real_4 (Skills of leaving employee: 1,2,9) | | | | |
|---|---|---|---|---|
| Skill's Proficiency | Dur = same, Cost = same | Dur = same, Cost = 2% | Dur = 5 days, Cost = same | Dur = 5 days, Cost = 0.6% |
| 0.1, 0.1, 0.1 | No solution found | solution found | No solution found | solution found |
| Real_5 (Skills of leaving employee: 8,9) | | | | |
| Skill's Proficiency | Dur = same, Cost = 5% | Dur = same, Cost = 15% | Dur = 5 days, Cost = same | Dur = 6 days, Cost = 13% |
| 0.1, 0.1 | No solution found | solution found | No solution found | solution found |
| Real_6 (Skills of leaving employee: 1,2,4,5,9) | | | | |
| Skill's Proficiency | Dur = same, Cost = same | Dur = same, Cost = 23% | Dur = 6 days, Cost = same | Dur = 6 days, Cost = 8% |
| 2, 2, 2, 2, 0.1 | No solution found | solution found | No solution found | solution found |

two (SK: 8, 9) and one (SK: 9) missing skill respectively. In this part of experiments, we allow shuffling of one to two employees to avoid shuffling people between teams and keep the teams stable [42].

We present results for all the three instances in Table 3.6. We start with same duration and cost values and increase step-wise. Our objective is to find the minimum proficiency value for each missing skill that new employee could have. The reason is that an employee with exactly the same skills' capability level, as resigning employee, may not be readily available in the market. For Real_4, our algorithm could not find any solution within the same duration and budget. However, a solution is found with a small increase of cost while keeping the duration the same. Another solution represents both an increase in duration and cost values. Real_5 has similar results to Real_4, i.e., no solution is found within the same duration and budget. Further, there is significant increase in the cost value to find an employee with minimum skill's proficiency level. Real_6 results reveal the fact that cost is increased significantly, if we keep the duration same and the missing skill's proficiency level to minimum. It also shows a trade-off between competing objectives that if we increase the duration then cost is reduced. This instance requires an employee with higher proficiency values to meet the threshold. It could be due to the reason that this instance has 5 employees to work on 91 tasks.

## 3.6 Summary

Software project scheduling under a dynamic environment is an important software engineering challenge. During the software development life cycle many unforeseen events may occur. In this context, 'employee turnover' is one of those volatile parameters which greatly influences the project execution. In this chapter, a model along

with an evolutionary approach which uses domain knowledge to generate the initial population, is introduced to deal with 'employee turnover' as a new dynamic event. Employee resignations may present big challenges for software companies and significantly impact the project schedule and cost if not handled well. When an employee has left, we first evaluate if rescheduling the remaining team can meet the manager's requirements based on the given project budget; if not, then we attempt to provide the manager with a set of options about what kind of person he/she should recruit to remain within the project budget.

The systematic experiments were carried out on benchmark problem instances and real data. The state-of-the-art and baseline algorithms were tested with these instances to see if they could handle this dynamic event. From the results, it has been observed that they are not capable of tackling 'employee turnover' dynamic events (part of RQ1 (section 1.4) in Chapter 1). A further study using our proposed approach reveals that rescheduling of existing employees increases project duration. Moreover, the experimental results have also shown the minimum skill level of new hired employee needed to remain within the budget (RQ2 (section 1.4) in Chapter 1).

# Chapter 4

# Onboarding a New Employee Under Dynamic Software Project Scheduling

In today's competitive and fastest growing market, software project managers are under increasing pressure to deliver quality software on time and within budget. Moreover, the development of a software project is associated with many unforeseen dynamic events which may impact development process, resulting in project failure if not handled well. For a successful project, an effective software project schedule needs to be adopted [116, 145]. The software project scheduling (SPS) problem deals with appropriate allocation of employees to tasks in a software project. Several studies show researchers' efforts to apply metaheuristic techniques to solve this problem.

This chapter aims at modelling the software project scheduling problem in a dynamic and uncertain environment with the consideration of multiple objectives namely, duration, cost, robustness, and stability along with variety of practical constraints. In SPS, adding a new employee into a running project is not unusual since software companies may need to replace a leaving team member or to augment the team's capacity, to speed up the project. In this regard, this chapter proposes a new heuristic approach to deal with 'new employee addition' dynamic events which has not been considered

before in the literature. Heuristics are usually designed for particular problem or even particular type of problem instances. They are problem-dependent and search for near-optimal solutions which signifies that the potential solution will be of good quality. The proposed heuristic method is based on the concept of $\epsilon$-MOEA [53] (see Chapter 2.1.2.2 for details) and uses domain knowledge to initialize the evolutionary algorithm's population, that generates a robust schedule. The reason for employing domain knowledge is that population initialization is a crucial task in evolutionary algorithms because the quality of the final solution and convergence speed can be affected [126].

The remainder of this chapter is structured as follows. Section 4.1 presents the motivation of this work. The proposed approach is described in Section 4.2. In Section 4.3, we define the problem objectives and constraints. Section 4.4 is devoted to experimental results. Finally, Section 4.5 presents conclusions.

## 4.1 Motivation

In general, project planning involves defining project scope, tasks, setting objectives, identifying deliverables, cost estimation, budget, and establishing a sequence of activities to further develop a schedule. Scheduling is categorized as one of the key problems in software project planning, and can be defined as the appropriate resource allocation and scheduling activities on time, along with optimizing the project objectives (e.g., duration and cost). A software project usually evolves under a dynamic environment where many uncertain events may occur, e.g., new employee addition, employee resignation, and task effort uncertainty. Such a situation makes the SPS problem NP-hard and challenging for software project managers. Traditional software project scheduling approaches consider a completely deterministic environment with very little place for uncertainty or dynamic events. A crucial deficiency with traditional planning methods is that they give more emphasis to deterministic environments with no occurrence of uncertain events that may result in late software deliveries, high development and maintenance costs, and unsatisfied sponsors. For instance, they assume that task effort and the skills possessed by each employee are known in advance. Further, it is

assumed that employees' skills do not change over time. However, in reality, a software project may face many kinds of hard-to-predict dynamic events that can disturb a software project plan, effecting project duration and budget. Like similar projects (e.g., civil engineering problems), the SPS problem can be solved using the critical path method (CPM) [70] and program evaluation and review technique (PERT) [161] but these methods may fail due to distinctive characteristics of software projects (such as nature of the team, application being built and customer). For example, they cannot effectively handle situations in which two or more projects share available resources, and require a lot of information to generate an effective project plan [70]. These aspects are handled by dynamic software project scheduling (DSPS) and make it more challenging.

Several studies have shown increasing interest in generating software project schedules with computer algorithms [9], but only a few researchers have addressed the disruptions during software development. Especially, no work considers 'new employee addition' dynamic event. Software organizations may hire a new employee to replace a leaving team member or to augment the team's capacity [22]. This new team member may also overcome the shortage of existing employees' skills demanded by current projects. Taking on an extra pair of hands can also increase the productivity. Moreover, in a real-world project, an automatic generation of improved software project schedule is helpful for project managers.

To address these, we propose a heuristic-based approach to deal with 'new employee addition' dynamic event. The proposed approach improves the efficiency of schedules along with robustness and stability objectives. The experimental results show that the proposed multi-objective evolutionary algorithm (MOEA) is capable of achieving promising solutions.

## 4.2 The Proposed Approach

This section describes a new heuristic method (h_NEA) to deal with a new dynamic event i.e. 'new employee addition', in order to address the shortcomings in an existing

state-of-the-art algorithm [139] (see Chapter 2.2.2 for details) that they are unable to handle other real-events. We make an assumption here that new employee is hired somewhere in the middle of project; however, Brook's law does not apply which states that adding manpower to a late project makes it later [21].

### 4.2.1 Heuristic Methods

The objective of the heuristic method is to produce near-optimal solutions in a reasonable time frame that solves the SPS problem for 'new employee addition' dynamic event. The SPS is an NP-hard combinatorial optimization problem [5]. Results about NP-hard problems make heuristics the only viable option for complex optimization problems that need to be solved in real-world applications [173]. These are designed for solving a problem where classic methods are too slow or may fail when finding an approximate solution.

The proposed heuristic (h_NEA) is divided into duration and cost heuristics. The purpose of the duration heuristic is appropriate allocation of a new employee to available critical tasks to reduce project duration whereas the cost heuristic allocates a new employee with an objective to minimize the project cost.

#### 4.2.1.1 Duration Heuristic

For new employee joining the software company, the purpose of duration heuristic is to reduce project duration. Detailed steps are given below:

**Step1**: Identify the critical path in the task precedence graph (TPG). The TPG indicates that a task should finish before starting a new preceding task. The complexity of each task is associated with it. The critical path (Figure 4.1) consists of all the tasks, in which any kind of task delay ultimately delays the whole project. This determines the shortest time possible to complete the project. Tasks in the critical path are dependent on each other with zero float [161].

**Step2**: The new employee is allocated to a critical task with the maximum dedication as long as his/her proficiency to that task is not zero. An employee's proficiency for a task is explained (see Chapter 3.4 for details) and calculated in Eq. (3.14).

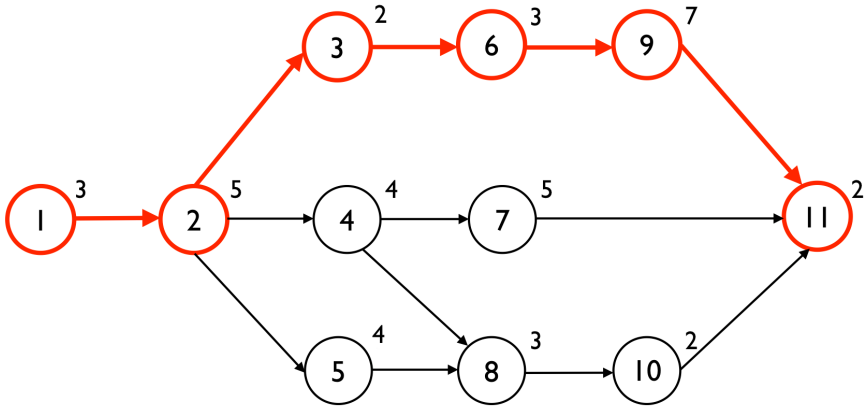



Figure 4.1: Example of Critical Path

Let us consider an example in Figure 4.1, where there are eleven tasks with corresponding task precedence graph and durations. Tasks $\{1, 2, 3, 6, 9, 11\}$ are critical tasks with zero float. The red color path shows the critical path. Hence, if the new employee is allocated to these tasks in the critical path with maximum dedication, it will reduce the project duration. For each critical task, a new heuristic solution is generated depending on new employee proficiency for that task. For duration heuristic, the 'task head count' constraint is checked at an algorithmic level. The pseudo code for the duration heuristic is given in Algorithm 2.

**Algorithm 2:** Duration Heuristic

**Input :** $e_n$: new employee, $T$: set of available tasks, $X$: dedication matrix, $G(T, A)$: Task precedence graph
**Output:** $D_{hs}$: Heuristic solutions with minimized project duration

```
for ∀T_j ∈ T do
    CT = critical tasks /* a subroutine findCriticalTasks() is called which finds
     all critical tasks using task precedence graph */
end
for ∀c_j ∈ CT do
    if e_nj^prof > 0 then
        X_ij : assign employee e_i to c_j with maximum dedication /* a heuristic
         solution is generated */
    end
end
return D_hs
```

#### 4.2.1.2 Cost Heuristic

This heuristic method (Algorithm 3) is designed in such a way to appropriately allocate a new employee to available tasks and reduce the project cost. According to [145], a software engineer's salary (a.k.a effort cost) is the main attribute that will impact on project cost. Besides, if the employee's proficiency is higher for a task, then the employee will finish that task in a shorter time. Therefore, we have considered three different cases to generate cost heuristic solutions as shown in Figure 4.2. For each available task, a new heuristic solution is generated depending on the cases stated below:

**Algorithm 3:** Cost Heuristic

**Input :** $E$: set of available employee, $e_n$: new employee, $T$: set of available tasks, $X$: dedication matrix

**Output:** $C_{hs}$: Heuristic solutions with minimized project cost

```
for ∀T_j ∈ T do
    for ∀e_i ∈ E do
        if e_n^prof > e_i^prof &e_n^salary < e_i^salary then
            X_ij ← 0
            if Task_skill_constraint(X_ij) = 0 then
                X_nj ← ∑ X_ij
            end
            if Employee_overwork_constraint(X_nj) > 0 then
                X_nj ← e_n^maxded
            end
        end
        if e_n^prof < e_i^prof &e_n^salary < e_i^salary then
            X_nj ← X_ij
            lowerTaskcost(e_n, e_i) /* a subroutine is called */
            return e_lower_cost
        end
        if e_n^prof > e_i^prof &e_n^salary > e_i^saalry then
            X_nj ← X_ij
            lowerTaskcost(e_n, e_i) /* a subroutine is called */
            return e_lower_cost
        end
    end
end
return C_hs
```

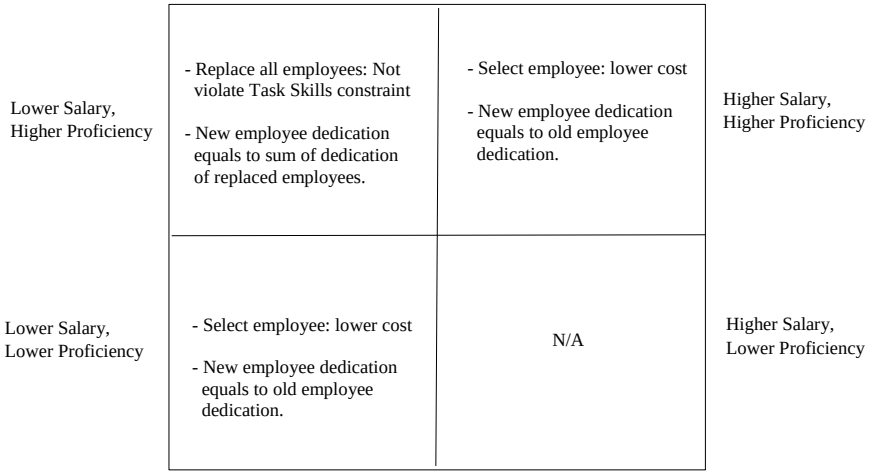



Figure 4.2: Cases for Cost Heuristic

- **Case a**. If the new employee's proficiency level is higher and salary is lower than all the existing employees, then replace all the employees who can be replaced (without violating task skills constraint) with new employee. The new employee's dedication is the sum of dedications of replaced employees. If new employee's dedication exceeds the employee's maximum dedication, then 'no employee overwork' constraint is considered here as well.

- **Case b**. If the new employee's proficiency level and salary is lower, then select between the existing and new employee who has lowest cost for performing a task. In this case, dedication for new employee and old employee are the same.

- **Case c**. If the new employee's proficiency level is higher and salary is also higher, then select between the existing and the new employee who has lowest cost for performing a task. In this case, the dedication for the new employee and the old employee is also the same.

For the cost heuristic, the new employee is replaced with the existing one if this replacement causes a significant difference in the project cost. It is benchmarked that this difference is 5% (experts subjective judgement) of the total cost. If difference is too small, there will be no benefit for replacement as new employee may not be proficient for that task as the old one.

### 4.2.2 MOEA-based Method for DSPS Problem

A set of Pareto-optimal solutions in a well-converged and well-distributed form, is an important issue in multi-objective evolutionary optimization. Many evolutionary algorithms do not use domain knowledge to generate the initial population. It is known fact that good initial estimates may generate better solutions with faster convergence depending on the prior knowledge existence, or if it can be generated at a low computational cost [126]. Therefore, in our approach, we design a heuristic method (h_NEA) using domain knowledge for population initialization. It is based on the concept of $\epsilon$-dominance [53] to deal with 'new employee addition' as new dynamic event. The procedure of the algorithm at scheduling point $t'$ is given in Figure 4.3. The initial population is formed using the following combination of solutions in step 1 of the algorithm.

$$X\%Heuristic + Y\%Baseline + Z\%Random \tag{4.1}$$

In Eq. (4.1), 50% are duration heuristic and cost heuristic solutions produced by Algorithm 2 and Algorithm 3 respectively; and its variants using mutation operator. 1% is pre-schedule/baseline/history solutions and 49% are random solutions. We are using trade-off between random initialization and the heuristic procedure. Random initialization will give diversification on the generated solutions while heuristics/baseline solution will converge the solutions rapidly. At initial time $t_0$, when the project starts, the initial population is generated using 100% random solutions in the initial step of algorithm and only duration, cost, and robustness as project objectives are solved.

## 4.3 Optimized Objectives and Constraints

### 4.3.1 Objective Functions

In this DSPS optimization problem, two objectives are considered as duration and cost to see how our proposed approach (h_NEA) works for new dynamic events as compared to current state-of-the-art algorithm [139] (see Chapter 2.2.2 for details). The mathematical formulation and detail of each objective is according to [139].

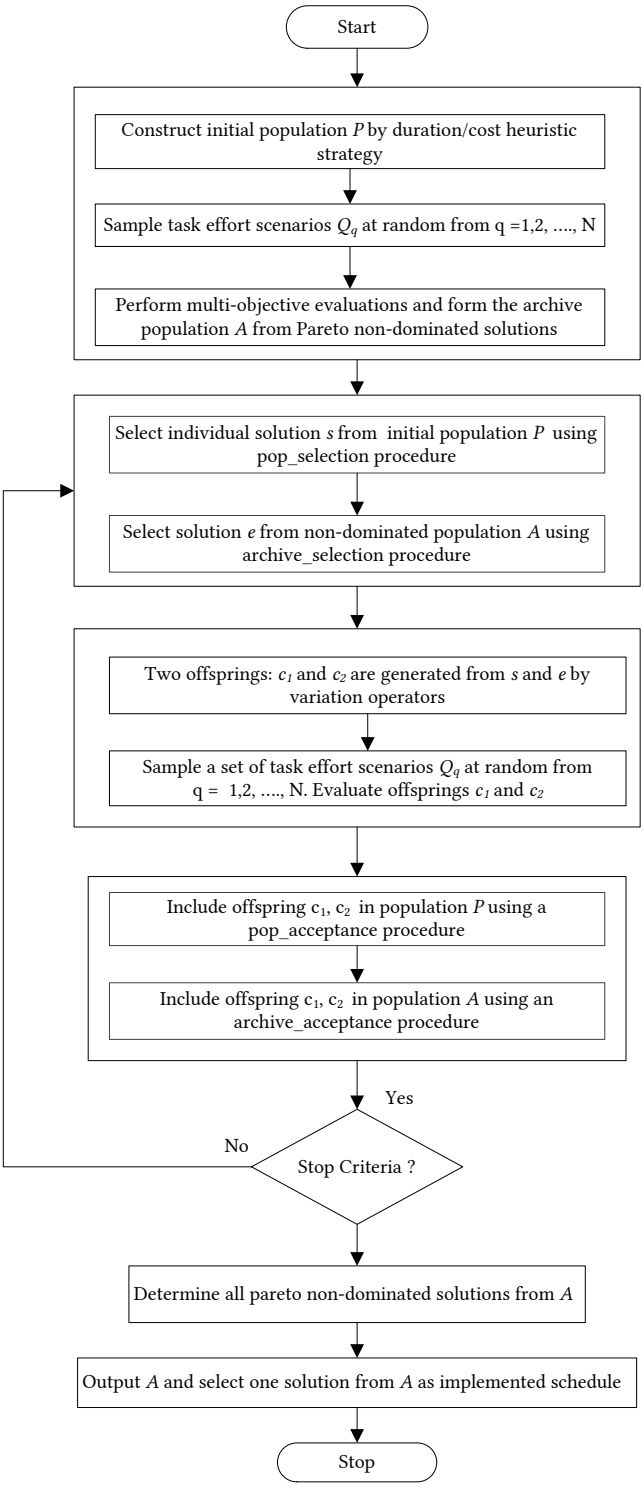



Figure 4.3: Procedure of algorithm at scheduling point '$t$''

$$\min \mathbf{F(t)} = [f_1, f_2, f_3, f_4] \tag{4.2}$$

where $f_1, f_2, f_3, f_4$ denotes duration, cost, robustness, and stability as project objectives respectively.

Duration. Project duration is time required to complete the project and is taken as the maximum finishing time of the last task.

$$f_1(t) = duration_I = max(T_j^{end}) - min(T_j^{start}) \tag{4.3}$$

$T_j^{end}$ and $T_j^{start}$ denotes the starting and ending time of each task. The end times for tasks without precedence are computed by adding the start time to respective duration of each task. The start time of a task with precedence is the end time of longest corresponding preceding task. The dependency between tasks is represented using task precedence graph (TPG). Moreover, duration for the whole project is the maximum finishing time of last task as calculated in Eq. (4.3).

Cost. Project cost is the total expenses required for the project completion.

$$f_2(t) = cost_I = \sum_{e_i \in E\_ava\_set} e\_cost_i \tag{4.4}$$

In Eq. (4.4), $E\_ava\_set$ is the set of available employees. Suppose an employee $e_i$ is assigned to a task $t_j$ with $X_{ij}$ dedication. The employee's normal monthly salary is $e_i^{norm\_salary}$. If an employee's dedication $X_{ij}$ is greater than 1, it indicates that he/she overworks for the project and overtime salary $e_i^{overwork\_salary}$ is also added to total salary. The expenses paid to an employee $e_i$ at $t'$ month are calculated as follows:

$if \sum_{j \in T\_active\_set} X_{ij} \leq 1.0$

$$e\_cost_i = e_i^{norm\_salary} \cdot t' \cdot \sum_{j \in T\_active\_set} X_{ij} \tag{4.5}$$

$if \sum_{j \in T\_active\_set} X_{ij} \geq 1.0$

$$e\_cost_i = e_i^{norm\_salary} \cdot t' \cdot 1 + e_i^{overwork\_salary} \cdot t' \cdot \left( \sum_{j \in T\_active\_set} X_{ij} - 1 \right) \tag{4.6}$$

In Eqs. (4.5) and (4.6), $T\_active\_set$ represents the set of active tasks which are being developed at time moment $t'$, where $t'$ denotes the current project development month. At time $t'$, a task that does not have any preceding unfinished task is known as active task.

Further, in Eqs. (4.3) and (4.4), subscript $I$ represents initial scenario without any uncertainty in tasks efforts. It means that initially estimated tasks efforts are correct and no task effort variance occurs.

To analyse the effect of our approach to other objective values, we then consider four objectives including robustness and stability. Improvement of duration and cost objectives by our approach may improve/deteriorate other objectives.

Robustness. It is the schedule's ability to cope with the task effort uncertainties for software project.

$$f_3(t) = \sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(max\left(0, \frac{duration_q(t') - duration_I(t')}{duration_I(t')}\right)\right)^2} + \sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(max\left(0, \frac{cost_q(t') - cost_I(t')}{cost_I(t')}\right)\right)^2} \tag{4.7}$$

In Eq. (4.7), $duration_I$ and $cost_I$ are initial duration and cost calculated in Eqs. (4.3) and (4.4) respectively which assumes that no task effort variance occurs. To cope with task effort uncertainties, a scenario-based approach is used. We sample a set of task efforts scenarios $\{\ Q_q | q = 1, 2, 3, ..N\ \}$ , where $N$ is sample size. In our case, $N = 30$. $duration_q$ and $cost_q$ are efficiency objectives under value $Q_q$. A 'max' function is used to truncate the variances of duration and cost decreases from initial value. The sampled effort for task $T_j^{effort_q}$ is calculated using the normal distribution [137].

$$T_j^{effort_q} = T_j^{effort_{mean}} + T_j^{effort_{varaince}} * z \tag{4.8}$$

In Eq. (4.8), '$z$' is a random scalar drawn from the normal distribution. At scheduling point $t'$, $T_j^{effort_q}$ is calculated multiple times until the condition $T_j^{effort_q} > T_j^{finished\_effort}$ is satisfied and then set $Q_q = \{T_j^{rem\_effort_q} | T_j^{rem effort_q} = T_j^{effort_q} -$

$T_j^{finished\_effort}\}$ where $T_j^{rem\_effort_q}$ means the $q_{th}$ sampled remaining effort of $T_j$ at scheduling point $t'$.

Stability. The purpose of this objective is to measure the deviation between new and pre-schedule. $t'$ represents the time when a dynamic event occurs and $t$ indicates pre-schedule time. It ensures that at each scheduling point, there is not too much variation in employees' assignment as compared to pre-schedule.

$$f_4(t) = stability = \sum_{i=1}^{E} \sum_{j=1}^{T} |X_{ij}(t') - X_{ij}(t)| * Penalty_{ij} \tag{4.9}$$

Where the values of $Penalty_{ij}$ are set as follows:

$$\begin{cases} 2, & \text{if } X_{ij}(t') > 0 \text{ and } X_{ij}(t) = 0, \\ 1.5, & \text{if } X_{ij}(t') = 0 \text{ and } X_{ij}(t) > 0, \\ 1, & \text{else.} \end{cases} \tag{4.10}$$

These penalty values are according to [139]. In first case, a large penalty value of '2' is given if employee $e_i$ performs a new task $t_j$ at scheduling point $t'$. He/she may need additional time to know about task specifications. In such a scenario, the employee's efficiency level to do work may be decreased. In the second case, a medium penalty of '1.5' is given if employee is not allocated to the same task for which he was working in pre-schedule. The employee might have received training and such training would be fruitless if the employee is not performing that task anymore. A small penalty of '1' is given if a task is performed with a different dedication level.

### 4.3.2 Constraints

The DSPS problem is subject to following constraints [139]. Below (i)-(iii) are hard constraints while (iv) is a soft constraint.

i. All tasks allocation constraint. Each task should be allocated to at least one

available employee.

$$\forall e_i \in \text{e \_available \_set(t)}, \sum T_j \text{\_Dedication} \neq 0 \tag{4.11}$$

ii. Task skills constraint. All the available employees allocated on a task must collectively cover all the skills required by that task.

$$T_j \text{\_Skills} \subseteq \cup e_i \ \{ \text{ Skills} — X_{ij} > 0\} \tag{4.12}$$

iii. No overwork constraint. No employee overworks for a project means that employee should not work for the project more than his/her maximum dedication.

$$\sum_{j \in T} X_{ij} \leq e_i^{max\_ded} \tag{4.13}$$

iv. Task head count constraint. According to the best practices of software engineering development, the number of employees assigned on a task should not exceed a limit. Each task has a maximum number of employees for our DSPS problem. $T_j^{max\_headcount}$ is calculated using the formula in [19].

$$\forall T_j, T_j^{no\_of\_emp} \leq T_j^{max\_headcount} \tag{4.14}$$

## 4.4 Experimental Results

This section answers the following research question by empirically investigating the proposed method:

- RQ1. Can existing methods deal efficiently with the 'new employee addition' dynamic events?
- RQ2. Does the proposed approach generate effective project plans in terms of duration and cost when dealing with 'new employee addition' dynamic events in contrast to existing approaches?

- RQ3. Does the improvement on objectives (duration and cost) made by the proposed heuristics compromise other objectives, e.g., project robustness and stability?

### 4.4.1 Experimental Setup

#### DSPS Instances

In this study, we use 18 dynamic benchmark and 3 real-world data instances.

***Benchmark.*** These 18 dynamic instances are derived from Alba and Chicano's benchmark [5] which are gathered from different software projects. The reason for choosing this benchmark data set is that these instances include variants of three important factors (number of employees, number of tasks and number of employee skills) for the DSPS problem as in real-world scenarios. To induce more reality, the dynamic data instances differentiate themselves from the static ones in [5] with the following keys aspects: task maximum headcount, task effort uncertainties, part-time jobs, and overworking of employees. In the project, it is assumed that part-time employees are 20 percent of the total employees whose maximum dedications are in the interval [0.5, 1); employees doing overtime work are 20 percent, whose maximum dedications are generated uniformly from (1, 1.5] at random; and remaining employees are full time, their maximum dedication is set to 1.0. If an employee possesses a skill, then relevant proficiency score for that skill is sampled uniformly from (0, 5]. If employee does not have any specific skill, then proficiency score is set to 0 for that skill. Following the practice in [5], an employee's normal monthly salary is sampled from a normal distribution with the mean of 10,000 and standard deviation of 1,000.

Further, task efforts variances are assumed to follow a normal distribution. These task efforts are calculated depending on different values of mean and standard deviation; and vary uniformly in interval [8,12] and [4,6] respectively. On average, the mean of a task effort is 10 and the standard deviation is 5 [139].

***Real-world.*** Three real-world instances derived from business software construction projects for a departmental store [70] are also used in our experiments.

The data instances are denoted as 'T30_E5_SK6-7', whereas the notations 'T30' and 'E5' represent total number of tasks and number of employees respectively. 'SK6-7' means number of skills possessed by an employee in the project. The 3 real-world instances are named as Real_1, Real_2 and Real_3.

### Parameter Settings

A set of parameter values for a rational comparison among algorithms are presented in Table 4.1.

Table 4.1: Parametrization (L = Individual Length)

| **State-of-the-art algorithm [139]** | |
|---|---|
| Population size | 100 individuals |
| Crossover | SBX, pc = 0.9 |
| Mutation | polynomial, pm = 1.0 / L |
| No of evaluations | 10,000 |
| **Proposed approach (h_NEA)** | |
| Population size | 100 individuals |
| Crossover | SBX, pc = 0.9 |
| Mutation | polynomial, pm = 1.0 / L |
| No of evaluations | 10,000 |

In this chapter, for each algorithm on each problem instance (18 benchmark and 3 real-world), 30 independent runs were executed to obtain all the results with the termination criterion of 10,000 evaluations.

### Evaluation of Solution Quality

In this study, the hypervolume (HV) indicator (see Chapter 2.1.2.3 for details) is used to compare algorithms' performance. HV is used for the problems whose Pareto fronts are unknown. Algorithms with higher HV values are desired. Here, we use a normalised HV value (also called hypervolume ratio (HVR)). With this normalisation, the range of all the obtained results is [0, 1], where value 1 represents the optimal value. In the calculation of HV, the reference point determination is a crucial issue. To calculate the reference point, we extract the maximal values for all objectives from the Pareto approximations found by all algorithms (used in comparison).

### 4.4.2 Performance Analysis of Existing Algorithms

This section analyses whether existing algorithms can handle 'new employee addition' dynamic events efficiently. Both state-of-the-art (SOA)[139] (see Chapter 2.2.2 for details) and baseline algorithm for new dynamic events are investigated. As stated before, the baseline algorithm is a MOEA-based complete rescheduling method [119] and generates initial population randomly at each scheduling point. It implies that new schedule is generated from scratch. The results are depicted by a clustered stacked graph in Figure 4.4. One column in the graph represents one objective for one data instance. The first objective is cost and$2^{nd}$ column represents duration. The objective values are normalized to be on the same scale. It is obvious that there is no clear pattern of results for both algorithms to deal with new dynamic events. For some instances, the baseline outperforms the state-of-the-art algorithm and vice versa. Furthermore, the objective values against data instances returned from both algorithms for 'new employee addition' dynamic event are presented in Table 4.2. It is not clear from results that which algorithm performs better for both duration and cost objectives. All these key factors necessitate the need for a new approach to deal with this event.

Figure 4.4: State-of-the-art and Baseline algorithm performance for 'New Employee Addition' dynamic event

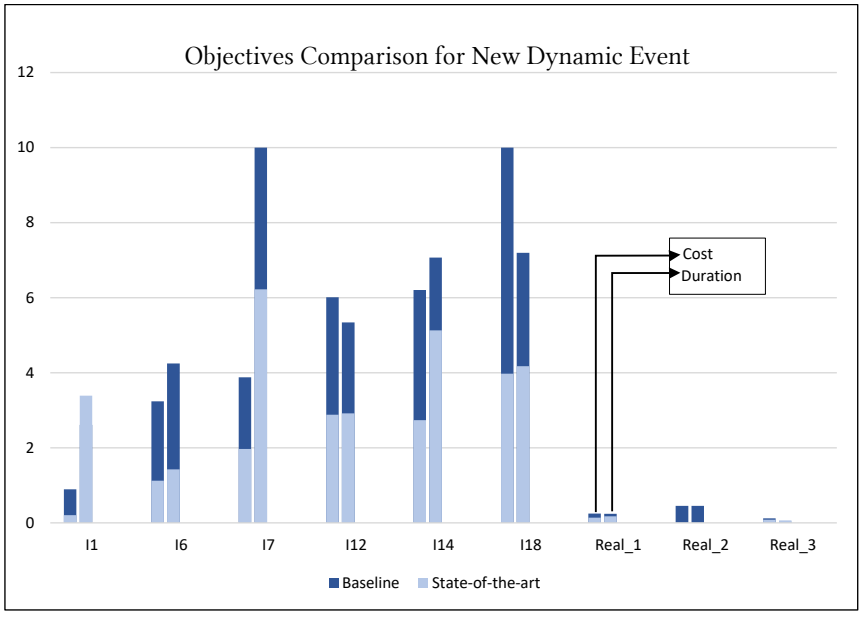


Table 4.2: Duration and Cost Values Comparison for Data Instances. The selected solution against each instance is according to the Decision Maker result [138].

| | Real_1 | | Real_2 | | Real_3 | |
|---|---|---|---|---|---|---|
| | SOA | Baseline | SOA | Baseline | SOA | Baseline |
| Duration | **4.57E+00** | 5.06E+00 | **3.31E+00** | 4.53E+00 | 3.80E+00 | **3.56E+00** |
| Cost | **1.15E+05** | 1.72E+05 | 3.55E+04 | **2.84E+04** | **8.40E+04** | 1.02E+05 |
| | T10_E5_SK4-5 | | T20_E10_SK4-5 | | T30_E10_SK4-5 | |
| | SOA | Baseline | SOA | Baseline | SOA | Baseline |
| Duration | 2.72E+01 | **2.17E+01** | **4.72E+01** | 7.38E+01 | **3.95E+01** | 5.32E+01 |
| Cost | **1.51E+05** | 5.26E+05 | **1.11E+06** | 2.16E+06 | **1.53E+06** | 3.43E+06 |

Evolutionary algorithms (EAs) provide a set of non-dominated solutions to solve any NP-hard problem [35]. Therefore, only single solution among feasible solutions is presented, according to the decision maker (DM) choice [138] (see Chapter 2.1.2.3 for details). In practice, a non-dominated solutions set found by EAs is provided to the software project manager subject to manager's choice. The project manager may select a solution considering best duration, best cost or best trade-off among all objectives. However, the involvement of a person for taking decisions is not practical in our experiments; hence, an automated decision making method [138] is adopted.

### 4.4.3 Comparison of Proposed Heuristic

The purpose of this comparison is to investigate that how significantly our proposed heuristic (h_NEA) deals with 'new employee addition' dynamic event. To show the effectiveness of our proposed approach, we present heuristic solutions produced by h_NEA for Real_1 and Real_2 data instances. We compare it with a state-of-the-art algorithm [139] using two objectives duration and cost. For a fair comparison, the baseline/history solution (implemented schedule) for both algorithms is the same. For h_NEA, at scheduling point $t'$, the population is composed of 1% history/baseline solution (implemented schedule), 50% heuristic solutions and 49% random solutions. For state-of-the-art algorithm, the distribution is 1% history/baseline solution, 50% history variants and 49% random solutions.

The duration heuristic optimizes and reduces the duration. The purpose of the cost heuristic is to minimize project cost by appropriate allocation of employees to tasks.

Table 4.3: Real_1 data instance: h_NEA vs Existing Algorithm Solutions

| h_NEA | | Existing Algorithm [139] | |
|---|---|---|---|
| Duration | Cost | Duration | Cost |
| History Solution | | History Solution | |
| 4.20E+00 | 1.69E+05 | 4.20E+00 | 1.69E+05 |
| Duration Heuristic Solutions | | Existing Algorithm Solutions | |
| **4.18E+00** | 1.70E+05 | 4.31E+00 | 1.68E+05 |
| **4.19E+00** | 1.66E+05 | 4.20E+00 | 1.69E+05 |
| **4.16E+00** | 1.67E+05 | 4.20E+00 | 1.69E+05 |
| Cost Heuristic Solution | | 4.20E+00 | 1.69E+05 |
| 4.32E+00 | **1.68E+05** | 4.19E+00 | 1.69E+05 |
| 4.20E+00 | **1.66E+05** | | |

Table 4.4: Real_2 data instance: h_NEA vs Existing Algorithm Solutions

| h_NEA | | Existing Algorithm [139] | |
|---|---|---|---|
| Duration | Cost | Duration | Cost |
| History Solution | | History Solution | |
| 2.44E+00 | 6.05E+04 | 2.44E+00 | 6.05E+04 |
| Duration Heuristic Solutions | | Existing Algorithm Solutions | |
| **2.42E+00** | 6.00E+04 | 2.29E+00 | 5.82E+04 |
| **2.39E+00** | 5.88E+04 | 2.58E+00 | 6.29E+04 |
| Cost Heuristic Solution | | 2.44E+00 | 6.05E+04 |
| 2.16E+00 | **5.88E+04** | 2.45E+00 | 6.06E+04 |
| 2.20E+00 | **5.74E+04** | 2.64E+00 | 6.38E+04 |
| 2.20E+00 | **5.65E+04** | | |

The results indicate (Table 4.3 and Table 4.4) that our approach significantly reduces both duration and cost objectives as compared to the existing one.

For the sake of simplicity, a Gantt chart is also drawn from the results of both algorithms in Figure 4.5 for the Real_3 data instance. The results indicate that the proposed method returns shorter project durations as compared to the state-of-the-art algorithm. Based on HVR values in Table 4.5, it is also obvious that h_NEA returns higher HVR value for 80% data instances and results are better.

Wilcoxon's rank sum tests [178] are also performed. It gives statically sound conclusion that results are significantly different from each other. "†" indicates that both algorithms are significantly different from each other at the 0.05 significance level.

Figure 4.5: Gantt Chart for Real_3: h_NEA vs Existing Algorithm. The selected solution is returned by Decision Maker result [138].

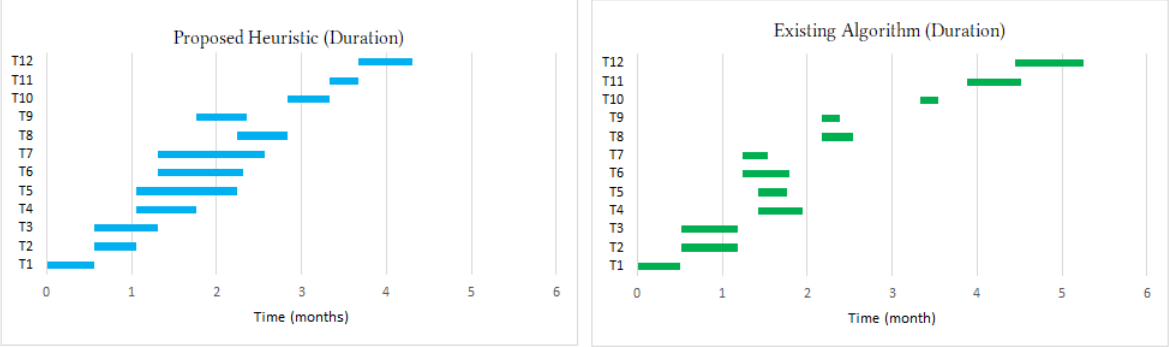



Table 4.5: Mean Value of HVR Indicator - 2 Objectives, best mean is in **boldface**.

| Instance | h-NEA | Existing Algorithm [139] |
|---|---|---|
| T10_E5_SK4-5 | **3.99E-01** † | 3.81E-01 |
| T10_E10_SK4-5 | **2.33E-01**† | 2.09E-01 |
| T10_E15_SK4-5 | **5.22E-01**† | 3.76E-01 |
| T10_E5_SK6-7 | 5.77E-01 † | **5.92E-01** |
| T10_E10_SK6-7 | **5.77E-01**† | 5.38E-01 |
| T10_E15_SK6-7 | **5.60E-01**† | 5.20E-01 |
| T20_E5_SK4-5 | **6.23E-01**† | 6.01E-01 |
| T20_E10_SK4-5 | **6.00E-01**† | 5.62E-01 |
| T20_E15_SK4-5 | **2.67E-01**† | 2.52E-01 |
| T20_E5_SK6-7 | **3.81E-01**† | 3.69E-01 |
| T20_E10_SK6-7 | 4.07E-01† | **4.10E-01** |
| T20_E15_SK6-7 | **1.77E-01**† | 1.70E-01 |
| T30_E5_SK4-5 | **5.31E-01**† | 4.92E-01 |
| T30_E10_SK4-5 | 5.06E-01† | **5.22E-01** |
| T30_E15_SK4-5 | 4.94E-01† | **5.07E-01** |
| T30_E5_SK6-7 | **4.98E-01**† | 4.07E-01 |
| T30_E10_SK6-7 | **5.55E-01**† | 5.51E-01 |
| T30_E15_SK6-7 | **5.18E-01**† | 4.96E-01 |
| Real_1 | **6.85E-01**† | 6.65E-01 |
| Real_2 | **4.23E-01**† | 3.80E-01 |
| Real_3 | **4.13E-01**† | 3.19E-01 |

"†" indicates that both algorithms are significantly different at significance level of 0.05 by Wilcoxon's rank sum test.

### 4.4.4 Effect on Other Objectives

This section explores whether proposed approach (h_NEA) affects the consideration of other two objectives (robustness and stability). It means that while improving duration and cost objective, other objectives are not deteriorated. To answer this, h_NEA is analysed using four project objectives as duration, cost, robustness, and stability. The Table 4.6 gives objective vector for data instances chosen randomly. It is clear from the result that robustness and stability objectives have also been improved along with duration and cost. This is due to the reason that robustness objective also has some dependency on duration and cost objectives and stability assures that there is not too much variation in employees' assignment. The proposed algorithm optimizes all objectives and selects a best trade-off among solutions. This analysis may be helpful for a manager with deeper insights about various trade-offs among multiple objectives.

To evaluate the effectiveness of our approach with four objectives, HVR quality indicator is also applied. In Table 4.7, proposed approach outperforms existing algorithm for 67% data instances including three real instances by returning higher HVR values.

Table 4.6: Objective Values Comparison for data instances. The selected solution is according to Decision Maker result [138].

| | Duration | Cost | Robustness | Stability |
|---|---|---|---|---|
| T10_E5_SK4-5 | | | | |
| h_NEA | **2.49E+01** | **5.03E+05** | **4.68E-01** | **6.17E-02** |
| SOA [139] | 2.50E+01 | 5.18E+05 | 5.43E-01 | 1.49E-01 |
| T20_E15_SK4-5 | | | | |
| h_NEA | **2.94E+01** | **8.57E+05** | **4.20E-01** | **3.70E+00** |
| SOA [139] | 3.60E+01 | 8.60E+05 | 4.92E-01 | 5.86E+00 |
| T30_E5_SK6-7 | | | | |
| h_NEA | **4.02E+01** | **1.40E+06** | **4.61E-01** | **6.38E-01** |
| SOA [139] | 7.33E+01 | 1.41E+06 | 6.70E-01 | 7.41E+00 |
| Real_3 | | | | |
| h_NEA | **3.95E+00** | **9.20E+04** | **8.18E-02** | **2.19E+00** |
| SOA [139] | 7.65E+00 | 9.54E+04 | 9.88E-02 | 2.45E+00 |

Table 4.7: Mean Value of HVR Indicator - 4 Objectives, best mean is in **boldface**.

| Instance | h-NEA | Existing Algorithm [139] |
|---|---|---|
| T10_E5_SK4-5 | **6.85E-01** † | 6.53E-01 |
| T10_E10_SK4-5 | 6.14E-01† | **6.19E-01** |
| T10_E15_SK4-5 | **6.43E-01** † | 6.25E-01 |
| T10_E5_SK6-7 | **7.09E-01**† | 7.07E-01 |
| T10_E10_SK6-7 | 6.24E-01 † | **6.36E-01** |
| T10_E15_SK6-7 | **6.76E-01** † | 6.28E-01 |
| T20_E5_SK4-5 | **7.49E-01** † | 7.39E-01 |
| T20_E10_SK4-5 | 7.38E-01 † | **7.73E-01** |
| T20_E15_SK4-5 | **7.07E-01** † | 6.95E-01 |
| T20_E5_SK6-7 | 6.56E-01 † | **6.66E-01** |
| T20_E10_SK6-7 | **7.31E-01** † | 7.21E-01 |
| T20_E15_SK6-7 | **7.19E-01** † | 6.85E-01 |
| T30_E5_SK4-5 | 7.17E-01 † | **7.47E-01** |
| T30_E10_SK4-5 | **6.89E-01** † | 6.75E-01 |
| T30_E15_SK4-5 | **7.33E-01** † | 7.18E-01 |
| T30_E5_SK6-7 | **6.87E-01** † | 6.74E-01 |
| T30_E10_SK6-7 | 7.03E-01 † | **7.18E-01** |
| T30_E15_SK6-7 | 7.32E-01 † | **7.55E-01** |
| Real_1 | **6.55E-01** † | 6.35E-01 |
| Real_2 | **6.46E-01** † | 5.47E-01 |
| Real_3 | **7.53E-01** † | 6.99E-01 |

"†" indicates that both algorithms are significantly different at significance level of 0.05 by Wilcoxon's rank sum test.

## 4.5 Summary

The need for efficiently allocating resources turn out to be increasingly important due to the development of complex software projects. This development is associated with a dynamic and ever-evolving environment with volatile project parameters. This is a critical issue in software engineering practices, as the total budget and human efforts must be managed well enough for successful project completion. This refers to a software project scheduling (SPS) problem under a dynamic environment which deals with employees to tasks allocations during the whole project development life cycle.

In this chapter, the SPS problem under a dynamic environment is dealt when a new employee joins the software organization. In SPS, adding a new employee into a running project is not unusual since software companies may need to replace a leaving

team member or to augment the team's capacity, to speed up the project. A heuristic-based approach is proposed which optimizes duration and cost objectives. It is based on the concept of $\epsilon$-MOEA [53] and uses domain knowledge to generate a robust schedule. The proposed algorithm, h_NEA, can mainly be characterised as:

- duration heuristic: finding the critical tasks using critical path method and allocating new employee to those tasks with the aim of reducing duration;
- cost heuristic: to minimize the cost, replace the old employee with new employee depending on employee's salary and proficiency level.

Systematic experiments were carried out on benchmark problem instances and real data. The state-of-the-art and baseline algorithms were tested with these instances to see if they could handle 'new employee addition' dynamic events. The results reveal that they could not efficiently handle this dynamic event (part of RQ1 (section 1.4) in Chapter 1). Moreover, h_NEA was compared against the peer algorithm [139]. The results show that our proposed approach is very competitive and achieves a good trade-off among convergence and diversity (RQ4 (section 1.4) in Chapter 1). It means that obtained solution set is well-converged and well-distributed in multi-objective optimisation problem. Further, we also demonstrate that other objectives (robustness and stability) have also been improved along with duration and cost objectives (RQ3 (section 1.4) in Chapter 1).

# Chapter 5

# Modelling Human Aspects for the Software Project Scheduling Problem

In this chapter, a new model for the software project scheduling (SPS) problem is presented, which takes into account an important human factor, i.e. employee experience. This model deals with two conflicting objectives as duration and cost. Moreover, the formulation of cost objective is based on real-world scenario. For the SPS problem, it is generally believed that i) human resources are the main resources [30] and ii) the human factors such as employee skills and experience, play a vital role in the success of software projects [175]. Inspired by these two facts, we present a new model for the SPS problem while incorporating important human factor 'employee experience'. The proposed model differs from existing models in such a way that it incorporates evolution of employees' experiences with their learning ability over time. Furthermore, we investigate how six state-of-the-art algorithms perform on this new model.

The remainder of this chapter is organised as follows. In Section 5.1, a brief introduction of state-of-the-model is given. Section 5.2, the motivation behind modelling human aspects for the SPS problem is presented. Section 5.3 is devoted to the presentation of proposed model which is followed by an overview of state-of-the-art algorithms

in Section 5.4. In Section 5.5, empirical results of proposed model in comparison with state-of-the-art model and further investigation of state-of-the-art algorithms performance on proposed model is carried out. Finally, Section 5.6 concludes this chapter.

## 5.1 Background

This section introduces the state-of-the-art model [139] for the SPS problem. The reason for selecting this model is that it is a dynamic version of the model presented in [28], and the dedication of each employee to each task is determined dynamically. The employee's and task's attributes for this model are defined in the next section with cited references. The mathematical model for two objectives is explained below:

$$\min \mathbf{F}(t) = [f_1, f_2] \tag{5.1}$$

where $f_1$, $f_2$ represents duration and cost objectives receptively.

$$f_1(t) = duration = max(T_j^{end}) - min(T_j^{start}) \tag{5.2}$$

The duration measure $f_1(t)$ in Eq. (5.2) is maximum elapsed time required for completion of each available task. $T_j^{end}$ and $T_j^{start}$ denotes the starting and ending time of each task. Moreover, duration for the whole project is the maximum finishing time of the last task.

$$f_2(t) = cost = \sum_{e_i \in E\_ava\_set} e\_cost_i \tag{5.3}$$

$f_2(t)$ in Eq. (5.3) represents project cost, which can be defined as the sum of all expenses payable to available employees against their dedications to project tasks. The set of available employees is denoted by $E\_ava\_set$. Let $T\_active\_set$ denotes the set of active tasks which are being developed at time moment $t'$, where $t'$ represents any month during which the project is being developed. A task is called active if it does not have any preceding unfinished task according to the TPG at time moment $t'$. Therefore, the expenses paid to the employee $e_i$ at $t'$ month are calculated as follows:

$if \sum_{j \in T\_active\_set} X_{ij} \leq 1.0$

$$e\_cost_i = e_i^{norm\_salary} \cdot t' \cdot \sum_{j \in T\_active\_set} X_{ij} \tag{5.4}$$

$if \sum_{j \in T\_active\_set} X_{ij} \geq 1.0$

$$e\_cost_i = e_i^{norm\_salary} \cdot t' \cdot 1 + e_i^{overwork\_salary} \cdot t' \cdot \left( \sum_{j \in T\_active\_set} X_{ij} - 1 \right) \tag{5.5}$$

where $e_i^{norm\_salary}$ is employee's normal salary, $e_i^{overwork\_salary}$ represents overwork salary, $X_{ij}$ is dedication of an employee $e_i$ to task $t_j$. $X_{ij}$ greater than 1 indicates that employee overworks for the project and overtime salary is also added to total salary.

## 5.2 Motivation

In the modern world, employee experience has been an increasingly important factor in human resource (HR) and business. A survey reports that 79% of business and HR leaders believe employee experience is very important trend and has become imperative for project success [13]. Moreover, an analysis of 250 global organizations reveals the fact that companies who have the highest score for employee experience generated two times highest average revenues, four times highest average profits and 40% lowest turnover [115]. Employee experience involves knowledge, skills, practice and situation familiarity. It promotes agility, service and rapid response to any software development activity, which are the most important drivers for success.

In the last two decades, the fast advancement in the software industry has confronted software houses with a competitive market. The success in such a highly competitive environment requires efficient and effective project plan to reduce the time and cost of quality software development that meet or exceeds customer's expectation [116]. This raises concern for the decisions who does what during the whole software project development life cycle, also known as software project scheduling (SPS) problem [5]. The SPS problem consists of allocating employees to tasks for a given project time-line. The classical methods like critical path method (CPM) [70], program evaluation and review technique (PERT) [161], and the resource-constrained project scheduling problem (RCPSP) model [83] have been intensively applied for solving the SPS problem

but they are becoming obsolete due to increasingly unique characteristics of software projects which are technology dependent. The failure of London Ambulance Service mega project was also the result of poor software project management [61]. Hence, it has escalated researchers' interest in developing new techniques where employees can be appropriately assigned to the tasks for project efficiency.

As software companies strive to get their jobs done faster with cost-cutting approaches in order to succeed [111], therefore, it has become important to consider about important factors into the model of SPS problem [137]. In this regard, prior research has found the existence of learning curves in manufacturing and service industries and provide evidence for positive impact on project scheduling [164]. However, none of existing studies considers and investigates the evolution of employees' experiences factor for the SPS problem. The software project development is a people intensive activity [33] that requires varying degree of skills and experience with newly emerging technologies. Therefore, we incorporate this important human attribute into the SPS model. This study provides insights into how significantly our proposed model reduces project duration and cost as compared to an existing state-of-the-art model [139]. In addition, the employee's experience significantly improves organization's productivity and is imperative for project success [115]. Bearing this in mind, a new model for the SPS problem with two objectives (project duration and cost) is presented.

## 5.3 The Proposed Model

This section presents a more practical version of mathematical model for the SPS problem, which considers human factors as increasing employee experience together with project duration and cost. Different from previous work [118], firstly, it incorporates employees' experiences with employees' learning ability over time. Secondly, the cost objective formulation is according to a real-world [146] situation. The employees and tasks are two major elements of the SPS problem instance as shown in Fig.2.1.1, their details are given below.

### 5.3.1 Employee's Attributes

Software development is a people-intensive activity. In real-world, a software house keeps record of each employee like employee's wages, skills and working constraints etc. Suppose 'n' employees are working for a software project as E $\{e_1, e_2, e_3, e_n\}$. For employee $e_i$, the attributes in Table 5.1 are considered.

Table 5.1: Employee's Attributes

| Symbol | Description |
|---|---|
| $e^{id}$ | Employee's specific id. |
| $e^{skills}$ [139] | The set of employee's skills in which he/she is proficient. |
| $e^{exp}$ | Employee's experience falls between [0,1], '1' refers to experienced employee, whereas '0' refers to the fresh employee. |
| $e^{basic\_salary}$ | Employee's basic salary per month. |
| $e^{perhour\_salary}$ | Employee's per hour salary. |
| $e^{overwork\_salary}$ [139] | Employee's overtime work salary. |
| $e^{nhours}$ | Employee's normal working hours per month. |
| $e^{available}$ [139] | Employee's availability during project. |
| $e^{max\_ded}$ [139] | The maximum dedication of employee $e_i$ for a software project represents the percentage of full-time job that he/she is able to dedicate. '$e^{max\_ded} = 1$' indicates that he/she spends all normal working hours on the project, whereas $e^{max\_ded} = 1.2$' means that he/she can work 20% more than normal working hours. |

### 5.3.2 Task's Attributes

A software project comprises of multiple tasks 'm' as T $\{t_1, t_2, ...., t_m\}$. For example, tasks could be interface design, business requirements, integrate system modules, candidate release, and production plan sign off etc. These tasks are executed according to a task precedence graph (TPG), which specifies which tasks should finish before starting a new preceding task. For task $t_j$, the following attributes are considered in Table 5.2.

Table 5.2: Task's Attributes

| Symbol | Description |
|---|---|
| $T^{id}$ | Task's specific id. |
| $T^{skills}$ [139] | The set of skills required to accomplish a task. |
| $T^{status}$ | Task status, '1' refers to active task whereas '0' refers to cancelled/suspended task. |
| $T^{prio}$ | Task's priority for execution as mentioned in task list {Very High, High, Medium}. |
| $T^{effort}$ [139] | Effort (unit: person-months) required to complete the task. |
| $T^{completed}$ | A binary variable indicating whether a task has been completed or not. '1' means task is unfinished and '$T_j^{completed} = 0$' shows task has been completed. |
| TPG [139] | TPG is an acyclic directed graph G(T,A) to represent the dependency between tasks. In TPG, the set of nodes represents the set of tasks T. The precedence relations among the tasks is denoted by set of arcs A. |

### 5.3.3 Objective Functions

To solve the SPS problem and provide software manager near-optimal project schedule, two project objectives namely, duration and cost are optimized.

$$\min \mathbf{F}(t) = [f_1, f_2] \tag{5.6}$$

where $f_1$, $f_2$ represents duration and cost objectives receptively.

**Duration** - It is the total time required to complete the project and is taken as the maximum finishing time of the last task.

$$f_1(t) = duration = max(T_j^{end}) - min(T_j^{start}) \tag{5.7}$$

The task duration is calculated in following way:

$$T_j^{dur} = \frac{T_j^{effort}}{\sum_{i \in E'} X_{ij} * (1 + k\varepsilon_t)} \tag{5.8}$$

$\varepsilon_t$ is assumed to be increased with the time $t$ as follows:

$$\varepsilon_t = \tanh(\varepsilon_o + a\triangle t) \tag{5.9}$$

In Eq. (5.7), $T_j^{end}$ and $T_j^{start}$ denotes the starting and ending time of each task. In Eq. (5.8), $T^{effort}$ is the total estimated effort for a task, $X_{ij}$ is employee to task dedication matrix, '$E'$' is number of employees, $\varepsilon_t$ is employee experience at time '$t$' and '$k$' is parameter for experience dependency of tasks i.e. how much time of task depends on experience. '$k$' range is between [0,1]. To represent employee's increasing experience over time, a sigmoid (hyperbolic tangent) function is used in Eq. (5.9) according to [137], where $\varepsilon_o$ is employee initial experience, '$a$' is growth rate of experience and we define $a = \frac{\mu}{\phi}$ where $\mu$ is employee learning factor between range [0.5,1.5]. $\phi$ is time unit parameter, it is defined as the number of time units in a month. Thus, if the time unit is a day, then $\phi = 30$. $\triangle t$ is time difference $t - t_o$ where '$t$' is highest value among task's predecessor durations and $t_o$ is initial time.

The employee's experience factor distinguishes this mathematical model from existing models. It is important to note that the employee's experience evolution is not based on the assumption that tasks are consistent that will improve employee's skills. In this model, we consider employee's diverse experience. It is independent regardless that he/she works on same tasks. Since software engineer's experience is difficult to measure [113], researchers often rely on two proxies for this abstract concept [141] i.e. length of experience and self-assessed expertise [8].

**Cost** - Project cost is the sum of all the expenses paid to all employees in the software development process until project completion.

$$f_2(t) = \sum_{i=1}^{E'} cost_i \tag{5.10}$$

In the real-world, there are two types of employees in a software house, permanent and temporary ones. An employee salary is divided into three parts: basic, per-hour normal working, and overwork salary [33]. Permanent employees have stable basic salaries every month without taking into account their dedication to the project. This basic salary is added to the per-hour salary which is based on the employee's normal working hours and his/her dedication to the project. The basic salaries are paid only to permanent employees while temporary employees have higher per-hour working salaries.

Suppose an employee $e_i$ dedicates $nhours$ to the project at $t'$ month with $X_{ij}$ dedication. The employee's basic salary $e_i^{basic\_salary}$ is added to the per-hour salary $e_i^{perhour\_salary}$ which is paid according to $nhours$ and $X_{ij}$ dedication. If employee's dedication ($X_{ij}$) is greater than 1, it indicates that employee overworks for the project and overtime salary $e_i^{overwork\_salary}$ is also added to the employee's total salary. The expenses paid to employee $e_i$ at $t'$ month are calculated as follows:

$if \sum_{j \in T} X_{ij} \leq 1.0$

$$cost_i = \left( \sum_{j \in T} X_{ij} * nhours * e_i^{perhour\_salary} \right) \cdot t' + e_i^{basic\_salary} \cdot t' \tag{5.11}$$

$if \sum_{j \in T} X_{ij} \geq 1.0$

$$\begin{aligned} cost_i = & \left( \sum_{j \in T} X_{ij} * nhours * e_i^{perhour\_salary} \right) \cdot t' + e_i^{basic\_salary} \cdot t' \\ & + \left( \left( \sum_{j \in T} X_{ij} * nhours \right) - nhours * e_i^{overwork\_salary} \right) \cdot t' \end{aligned} \tag{5.12}$$

In contrast to other objective functions in the SPS literature, the duration objective is based on an important human factor 'employee experience' whereas the cost objective is designed by considering normal working hours, basic salary, per-hour salary, and overwork salary as in real-world situation [146].

### 5.3.4 Constraints

In this work, we consider following two hard constraints.

i. All Tasks Allocated Constraint. All tasks should be allocated to the available employees.

$$\forall e_i \in e\_available\_set(t), \sum T_j\_Dedication \neq 0 \tag{5.13}$$

ii. Task Skills Constraint. All the allocated employees to the task must fulfil the skills required by that task.

$$T_j\_Skills \subseteq \cup e_i \{Skills | X_{ij} > 0\} \tag{5.14}$$

Regarding constraint handling, a solution is penalized by multiplying objective values with a high number if any constraint is violated. In our case, we are multiplying with 1,000.

In contrast to models introduced in Chapter 3 and Chapter 4, this model considers about important human factors i.e. employee experience and project cost calculation is done according to real-world situation.

## 5.4 Multi-objective Algorithms

Search Based Software Engineering (SBSE) [79], a sub-area of software engineering, applies search-based techniques to solve real-world large scale problems; for example, software effort estimation [112, 87, 68], software defect prediction [158, 159, 23], and software testing [148, 102, 174, 176]. The SPS is one of those problems for which evolutionary algorithms (EAs) have been widely applied [33]. These EAs are robust and solve real-world large scale problems efficiently by providing a set of non-dominated solutions known as Pareto optimal set [50]. The representation of the Pareto optimal set in the objective space is known as the Pareto front [50].

In recent years, the optimization problems involving two or more conflicting objectives have received researchers' increasing attention. These are referred as multi-objective optimization problems (MOPs) whereas search-based optimization algorithms for solving MOPs are known as multi-objective algorithms [41]. The main goal of MOPs is obtaining the Pareto front. There can be a large (possibly infinite) number of non-dominated solutions in Pareto front. In practice, a Pareto front is an approximation of a finite number of non-dominated solutions. In this context, this set of solutions must fulfil two features: convergence and diversity [50]. **Convergence** means that solutions should be as close as possible to the exact Pareto front. It ensures that we are dealing with good-enough solutions [50]. **Diversity** is good exploration of the search space that solutions are uniformly spread and no regions are left unexplored [50]. For example, for the SPS problem (where the software project manager desires a project schedule with both low project cost and smaller duration), there does not exist one single schedule that achieves both objectives.

The studies in [35, 44, 108, 110] analyse the performance of classical algorithms on the same model as in [5]. Therefore, we investigate six state-of-the-art multi-objective algorithms (NSGA-II [54], NSGA-III [51], Two_Arch2 [160], BCE [104], OMOSPO [142], SMPSO [117]) for our proposed model which includes an important human factor. These six multi-objective algorithms are capable of finding a set of trade-off solutions in one single run and solve MOPs in an optimal way. Therefore, they are also known

as evolutionary multi-objective (EMO) algorithms. The purpose of comparing these algorithms for the SPS problem is to provide the project manager the best algorithm which gives near-optimal project schedule. The reason for selecting these specific six algorithms is that they are simple to implement for the SPS problem and they are shown to be effective on other problems as mentioned earlier, for example, software effort estimation [112, 87, 68], software defect prediction [158, 159, 23], and software testing [148, 102, 174, 176]. Some of them (NSGA-II, NSGA-III, OMOSPO, SMPSO) also have existing implementation in the jMetal framework [59]. jMetal is an object-oriented Java-based framework aimed at the development, experimentation, and study of metaheuristics for solving MOPs.

The NSGA-II was introduced by Deb et al. [54]. A fast non-dominated sorting solution and the preservation of the solution's diversity are its two principal parts. NSGA-III [51] is an improved version of NSGA-II and is more suitable to deal with many objectives (more than three) problems. Two_Arch2 [160] is a many objective algorithm comprising of two archives focusing on convergence and diversity separately. The BCE [104] framework is composed of two parts: pareto criterion (PC) evolution and non-pareto criterion (NPC) evolution to deal with the MOPs. These two parts work collaboratively to facilitate each other's evolution. OMOPSO [142] is a multi-objective particle swarm optimization (MOPSO) [39] variant which includes an external archive and mutation operators to expedite the convergence of swarm. SMPSO [117] is also a MOPSO which uses a velocity constriction mechanism with the aim of enhancing the search capability. More details about these algorithms can be found in Chapter 2.1.2.2.

## 5.5 Experimental Results

This section answers the following research questions by empirically investigating the proposed method:

- RQ1: What benefit can the proposed model bring to the SPS problem?
- RQ2: How do state-of-the-art algorithms perform on proposed model? Do they perform similarly, or are specific algorithms particularly suitable for this model?

### 5.5.1 Experimental Setup

#### SPS Instances

In this section, the experiments are conducted on broad range of benchmark and real-world data instances.

***Real-world.*** The six real-world data instances are used in our experiments. The 3 data instances among these real instances have been derived from business software construction projects for a departmental store [70]. These 3 small real instances have 10 employees with 15 tasks maximum, and do not address inclusion of employee experience factor. For these instances, we have randomly generated values between [0,1] for employee experience attribute. To measure the scalability of our model, we also use 3 real instances obtained from a software company. These data instances comprise of 10,8,5 employees and 42,43, and 91 tasks respectively. For these data instances, employee's experience is measured by a software project manager based on two proxies as mentioned in [8] i.e. length of experience and self-assessed expertise.

***Benchmark.*** Benchmark instances are derived from Alba and Chicano's benchmark [5]. These specific 18 data instances include variants of three important parameters (number of employees, number of tasks and number of employee skills) for the SPS problem as in real-world scenario. To induce more reality, these instances differ from static instances [5] in the following keys aspects: task maximum headcount, task effort uncertainties, part-time jobs, overworking of employees, and employees experience. The data of 18 instances, derived from [5], are gathered from different software projects.

Each instance has different number of employees and tasks which can be (5, 10, or 15), and (10, 20, or 30) respectively. Each employee can possess different skills which ranges from 4 to 5, or from 6 to 7. The total number of different skills $SK$ is 10, and a task is associated with five skills randomly selected from them. In the project, it is assumed that part-time employees are 20 percent of the total employees whose maximum dedications are in the interval [0.5, 1); another 20 percent of employees do overtime work, whose maximum dedications are generated uniformly from (1, 1.5] at

random; and remaining employees are full time, their maximum dedication is set to 1.0. Each skill is associated with a proficiency score. The proficiency score for a skill is sampled uniformly from (0, 5] at random. If an employee does not have any specific skill, then proficiency score for that skill is set to 0. Further, an employee's normal monthly salary is sampled from a normal distribution with the mean of 10,000 and standard deviation of 1,000 following the practice in [5]. The employee overtime salary is the normal monthly salary multiplied by 3. For benchmark instances, employee's experience is generated randomly between [0,1].

The data instances are denoted as 'T20_E10_SK4-5', whereas 'T20' means total number of tasks, 'E10' represents total number of employees, and 'SK4-5' means number of skills for an employee. As an another example, T10_E5_SK6-7 represents that the project has 10 tasks with 5 employees, each employee with 6 or 7 skills. The 6 real instances are named as 'Real_n_Tx_Ey' where n $\epsilon$ {1,2,...6} and x, y represents number of tasks and number of employees respectively.

### Parameter Settings

A set of parameter values for a rational comparison among algorithms are presented in Table 5.3.

Table 5.3: Parametrization (L = Individual Length)

| **NSGA-II, NSGA-III, Two_Arch2, BCE** | |
|---|---|
| Population size | 100 individuals |
| Selection of Parents | binary tournament + binary tournament |
| Recombination | simulated binary, $p_c = 0.9$ |
| Mutation | polynomial, $p_m = 1.0$ / L |
| **OMOPSO** | |
| Swarm size | 100 particles |
| Mutation | uniform + non-uniform, $p_m = 1.0$ / L |
| Leader size | 100 |
| **SMPSO** | |
| Swarm size | 100 particles |
| Mutation | polynomial, $p_m = 1.0$ / L |
| Leader size | 100 |

#### Evaluation of Solution Quality

The hypervolume (HV) metric [179] is used to compare i) the performance of proposed model against state-of-the-art model [139] and ii) the state-of-the-art algorithms performance for the new model. Here, we use a normalised HV value (also called hypervolume ratio (HVR)). With this normalisation, the range of obtained results reside in $[0, 1]$, where 1 represents the optimal value. Usually, there are two crucial issues in the calculation of HV i.e. search space scaling [62] and the reference point choice [7, 63]. Since the objectives in the SPS problem reside in different ranges of values, therefore, the objective values of the final solution set is standardized with respect to the range of the obtained Pareto front. We set the reference point 1.1 times the upper bound of the Pareto front to balance between convergence and diversity of the obtained solution set, according to [89].

In this chapter, for each algorithm on each problem instance (18 benchmark and 3 real-world), 30 independent runs were executed to obtain all the results with the termination criterion of 10,000 evaluations. Furthermore, we set population size to 100 according to practice in [64, 157].

### 5.5.2 Comparison of Proposed SPS Model with State-of-the-art Model

In this section, the performance of proposed model against state-of-the-art model [139] using a baseline algorithm (NSGA-II [54]) is presented. The main purpose of this comparison is to validate that our proposed model impacts and reduces project duration and cost. To address this, we compare results of proposed model by those achieved by existing approach in Table 5.4. The results show that our proposed model significantly reduces project duration and cost for 6 real-world data instances. Since evolutionary algorithms provide a set of non-dominated solutions to solve any NP-hard problem, therefore, a non-dominated solution set is obtained for each problem instance. It also implies that the software project manager can choose from a set of solutions with a good balance between project cost and duration. We choose and present one solution using an automatic decision making method [139] (see Chapter 2.1.2.3 for more details).

Table 5.4: Performance Comparison of Proposed vs Existing Model. The selected solution against each instance is returned by Decision Maker result [138].

| | Proposed Model | | Existing Model | |
|---|---|---|---|---|
| | Duration | Cost | Duration | Cost |
| Real_1_T15_E10 | **3.63E+00** | **8.09E+04** | 4.48E+00 | 1.58E+05 |
| Real_2_T15_E10 | **2.66E+00** | **2.52E+04** | 8.28E+00 | 4.66E+04 |
| Real_3_T12_E10 | **1.91E+01** | **1.97E+04** | 2.90E+01 | 8.35E+04 |
| Real_4_T42_E10 | **1.42E+01** | **2.84E+06** | 2.52E+01 | 7.50E+06 |
| Real_5_T43_E8 | **4.26E+01** | **2.49E+06** | 5.27E+01 | 6.00E+06 |
| Real_6_T91_E5 | **1.71E+01** | **4.84E+06** | 2.28E+01 | 1.22E+07 |

To compare the performance using quality indicator, Table 5.5 shows the HVR values for both models for benchmark and real-world data instances. These values are average of 30 independent runs. Algorithms returning higher HVR values perform better [155]. For most of the benchmark data instances, our model returns higher HVR values. Moreover, existing model returns zero HVR value for five out of six real data instances, it means that no solutions are found which dominate the reference point. That's why it is clear from results that our novel formulation is more effective as compared to the existing one.

To further elaborate the effectiveness of our proposed model, we draw a Gantt chart for real-world instances obtained by the proposed and existing models in Figure 5.1. An experienced employee is able to finish a task earlier. It is obvious from the results that inclusion of an important human factor into SPS model returns a shorter project duration as desired.

### 5.5.3 Comparison of State-of-the-art Algorithms for Proposed Model

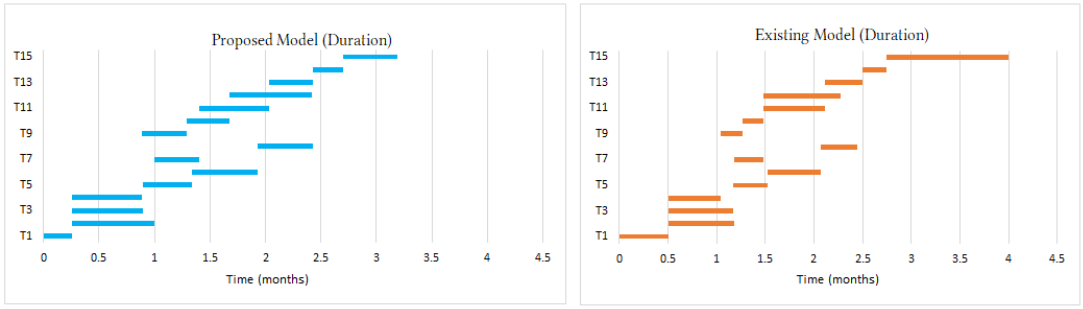
In this section, we analyse which state-of-the-art algorithm is best fit for our proposed model while considering employee's experience factor. As mentioned earlier, studies in [35, 44, 108, 110] analyse the performance of classical metaheuristic on the same model as proposed in [5]. The comparison of six targeted state-of-the-art algorithms based on HVR values is to identify which algorithm provides the software project managers near-optimal project schedules.

Table 5.5: HVR values obtained by NSGA-II on the Proposed and Existing Model, best mean is highlighted in **boldface**.

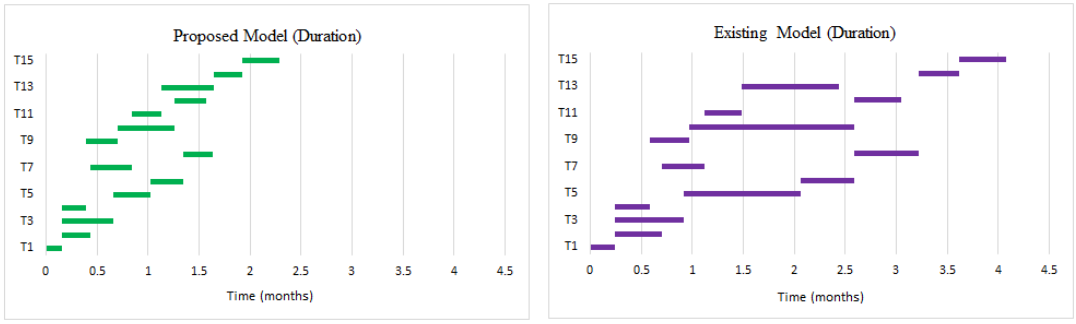
| | Proposed Model | Existing Model |
|---|---|---|
| T10_E5_SK4-5 | **1.90E+00** | 0.00E+00 |
| T10_E10_SK4-5 | 6.74E-01 | **9.73E-01** |
| T10_E15_SK4-5 | 7.23E-01 | **9.04E-01** |
| T10_E5_SK6-7 | **8.48E-01** | 0.00E+00 |
| T10_E10_SK6-7 | 7.90E-01 | **9.59E-01** |
| T10_E15_SK6-7 | 7.68E-01 | **9.48E-01** |
| T20_E5_SK4-5 | **5.03E-01** | 0.00E+00 |
| T20_E10_SK4-5 | **6.30E-01** | 5.19E-01 |
| T20_E15_SK4-5 | 6.12E-01 | **6.48E-01** |
| T20_E5_SK6-7 | **9.73E-01** | 1.12E-01 |
| T20_E10_SK6-7 | 5.24E-01 | **5.92E-01** |
| T20_E15_SK6-7 | **7.93E-01** | 3.23E-01 |
| T30_E5_SK4-5 | **1.21E-01** | 0.00E+00 |
| T30_E10_SK4-5 | **5.53E-01** | 4.75E-01 |
| T30_E15_SK4-5 | **5.90E-01** | 4.83E-01 |
| T30_E5_SK6-7 | 2.15E-01 | **7.00E-01** |
| T30_E10_SK6-7 | **5.29E-01** | 4.27E-01 |
| T30_E15_SK6-7 | **7.01E-01** | 3.21E-01 |
| Real_1_T15_E10 | **9.48E-01** | 0.00E+00 |
| Real_2_T15_E10 | **9.59E-01** | 5.75E-03 |
| Real_3_T12_E10 | **8.21E-01** | 0.00E+00 |
| Real_4_T42_E10 | **9.31E-01** | 0.00E+00 |
| Real_5_T43_E8 | **8.55E-01** | 0.00E+00 |
| Real_6_T91_E5 | **5.70E-01** | 0.00E+00 |

HVR results in Table 5.6 are the mean and standard deviation (SD) values. For each problem instance, the best mean values among the algorithms are shown with **bold** font. Moreover, concerning the statistically sound conclusions, the Wilcoxon's rank sum test is applied [178] (using the significance level of 0.05) to determine that if the results obtained by the state-of-the-art algorithms are significantly different from each other.

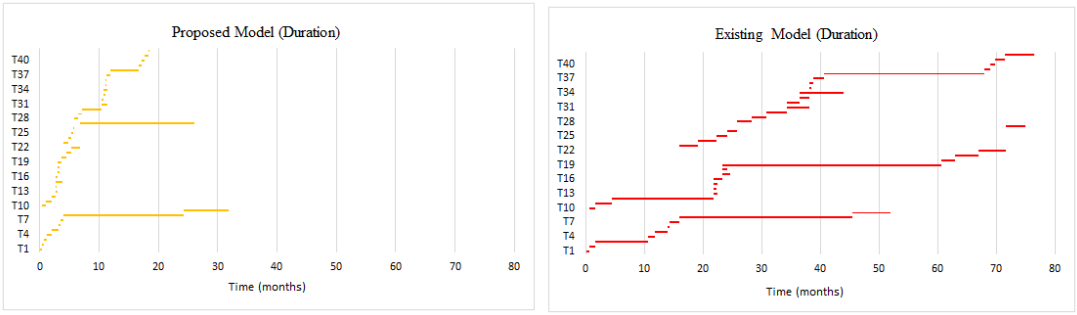
It can be observed from Table 5.6 that for most of data instances, BCE is giving better results based on HVR values. After BCE, NSGA-II, OMOPSO and Two_Arch2 are performing better for rest of the instances. The results also show that the state-of-the-art algorithms are significantly different from each other for most of the test data




(a) Gantt Chart for Real_1




(b) Gantt Chart for Real_2




(c) Gantt Chart for Real_4

Figure 5.1: Gantt Chart for Real instances: Proposed vs Existing Model. The selected solution is returned by Decision Maker result [138].

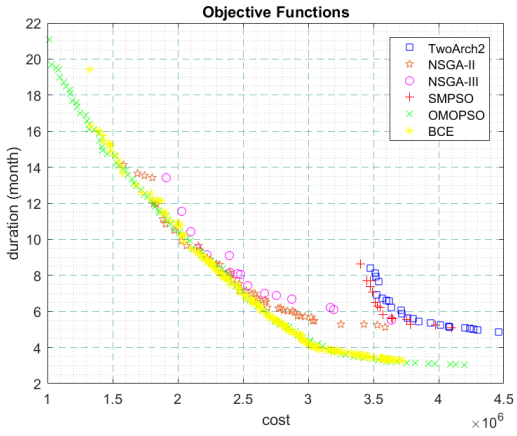
instances.

The final solutions of a single run of all state-of-the-art algorithms are plotted in Figure 5.2 regarding two-dimensional objective space $f_1$ and $f_2$, for benchmark data instances 'T10_E15_SK4-5', 'T10_E15_SK6-7', and 'T30_E15_SK4-5' as chosen randomly. As shown, BCE returns a well-converged and well-distributed set of solutions whereas

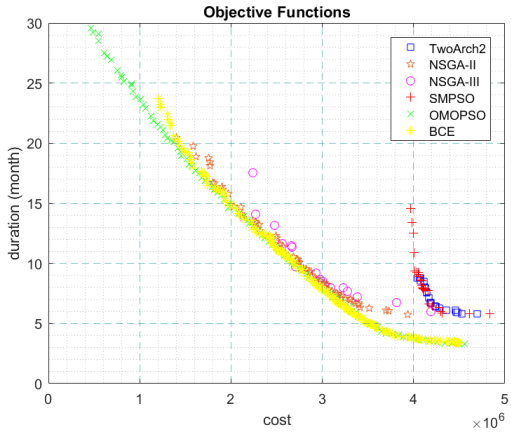
Table 5.6: Mean and Standard Deviation Values of HVR indicator, best mean is highlighted in **boldface**.

| | BCE | NSGA-II | NSGA-III | Two_Arch2 | OMOPSO | SMPSO |
|---|---|---|---|---|---|---|
| I1 | 0.00E+00(0.00E+00) | **8.87E-01**(4.78E-01) † | 2.69E-04(1.45E-3) † | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) |
| I2 | **8.81E-01**(1.32E-02)† | 8.24E-01(2.12E-02) † | 7.30E-01(2.57E-02)† | 5.51E-01(7.20E-03)† | 8.37E-01(3.61E-02) † | 5.62E-01(5.13E-03) † |
| I3 | **9.16E-01**(4.19E-02) † | 8.03E-01(2.32E-02) † | 7.05E-01(3.48E-02) † | 2.21E-01(7.32E-03) † | 7.91E-01(4.83E-02) † | 2.30E-01(6.62E-03) † |
| I4 | 0.00E+00(0.00E+00) | **2.91E-01**(4.57E-01) † | 2.30E-01(3.94E-01)† | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) |
| I5 | **8.91E-01**(2.05E-02) † | 8.39E-01(3.50E-02) † | 7.26E-01(3.58E-02) † | 5.13E-01(9.72E-03) † | 8.27E-01(3.86E-02) † | 5.25E-01(5.93E-03) † |
| I6 | **9.48E-01**(2.32E-02)† | 8.37E-01(2.25E-02) † | 7.34E-01(3.00E-02) † | 1.99E-01(8.11E-03) † | 8.96E-01(3.65E-02) † | 2.16E-01(8.64E-03) † |
| I7 | 0.00E+00(0.00E+00) | **1.15E-01**(7.90E-01) † | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) |
| I8 | **8.40E-01**(4.63E-02) † | 4.59E-01(3.48E-02) † | 2.98E-01(5.34E-02) † | 2.41E-01(2.22E-02)† | 8.14E-01(4.46E-02)† | 2.41E-01(3.34E-02) † |
| I9 | 6.44E-01(8.84E-02) † | 5.55E-01(4.20E-02) † | 3.08E-01(6.24E-02) † | 1.01E-01(1.78E-02) † | **6.87E-01**(6.99E-02) † | 8.33E-02(3.59E-02) † |
| I10 | **8.14E-01**(4.78E-02) † | 1.47E-01(2.94E-02) † | 6.22E-02(4.50E-02) † | 6.02E-02(2.15E-02)† | 7.05E-01(6.35E-02) † | 5.60E-02(2.62E-02) † |
| I11 | **7.96E-01**(7.49E-02)† | 4.69E-01(4.31E-02) † | 2.41E-01(8.77E-02) † | 2.26E-01(2.01E-02) † | 7.84E-01(5.20E-02) † | 2.11E-01(3.78E-02) † |
| I12 | 7.35E-01(5.09E-02) † | 7.36E-01(2.63E-02) † | 5.99E-01(5.09E-02) † | 4.46E-01(1.96E-02) † | **7.75E-01**(3.10E-02) † | 4.57E-01(2.34E-02) † |
| I13 | 9.86E-02(5.34E-02) † | 9.94E-02(5.26E-01) † | 9.77E-05(5.26E-04) † | **5.37E-01**(4.62E-02) † | 9.78E-02(3.49E-02) † | 5.09E-01(5.61E-02) † |
| I14 | 2.47E-01(5.40E-02) † | 1.27E-01(1.56E-02) † | 7.42E-02(2.50E-02) † | **8.98E-01**(2.54E-02) † | 2.73E-01(3.29E-02)† | 8.56E-01(2.64E-02) † |
| I15 | **8.60E-01**(4.72E-02) † | 4.75E-01(2.40E-02) † | 3.67E-01(4.15E-02) † | 2.69E-01(2.09E-02) † | 8.16E-01(3.77E-02) † | 2.63E-01(2.83E-02) † |
| I16 | **5.05E-01**(2.38E-01) † | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 8.44E-02(1.11E-01) † | 0.00E+00(0.00E+00) |
| I17 | **4.27E-01**(2.29E-01)† | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 0.00E+00(0.00E+00) | 2.62E-01(.42E-01) † | 0.00E+00(0.00E+00) |
| I18 | 6.83E-01(1.24E-01)† | 4.22E-01(3.76E-02) † | 2.33E-01(5.92E-02) † | 5.54E-02(2.28E-02) † | **6.85E-01**(7.59E-02)† | 4.43E-02(3.07E-02) † |
| Real_1 | **9.81E-01**(4.09E-03) † | 9.09E-01(1.41E-02) † | 8.58E-01(1.43E-02)† | 0.00E+00(0.00E+00) | 9.20E-01(2.17E-02) † | 0.00E+00(0.00E+00) |
| Real_2 | **9.67E-01**(3.13E-03)† | 9.43E-01(6.91E-03) † | 9.02E-01(6.56E-03) † | 4.51E-01(1.54E-02) † | 9.58E-01(4.71E-03) † | 5.08E-01(1.59E-02) † |
| Real_3 | 7.44E-01(4.56E-03) † | **7.89E-01**(7.59E-02) † | 7.23E-01(1.38E-02) † | 0.00E+00(0.00E+00) | 7.28E-01(7.19E-03) † | 0.00E+00(0.00E+00) |
| Real_4 | **9.73E-01**(9.74E-03) † | 8.76E-01(1.29E-02) † | 8.11E-01(1.88E-02) † | 0.00E+00(0.00E+00) | 9.26E-01(1.66E-02) † | 0.00E+00(0.00E+00) |
| Real_5 | **9.83E-01**(1.33E-03) † | 8.65E-01(6.09E-03) † | 8.48E-01(7.40E-03) † | 1.04E-01(1.66E-02) † | 9.73E-01(4.35E-03) † | 1.69E-01(1.10E-02) † |
| Real_6 | **8.73E-01**(3.71E-02) † | 2.12E-01(3.70E-02) † | 1.50E-01(5.24E-02) † | 1.56E-01(2.44E-02) † | 8.16E-01(3.22E-02) † | 2.33E-01(2.27E-02) † |

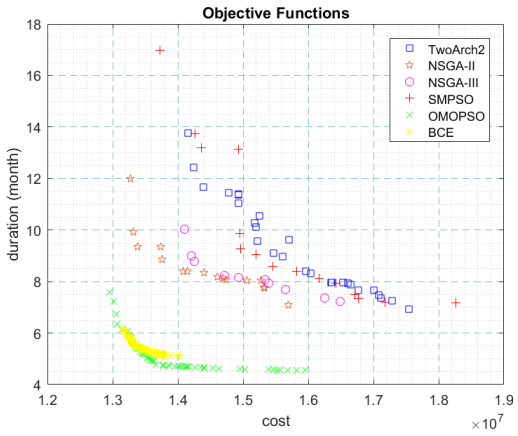
"†" indicates that each algorithm result is significantly different from each other at 0.05 significance level by the Wilcoxon's rank sum test.




(a) 'T10_E15_SK4-5' instance




(b) 'T10_E15_SK6-7' instance

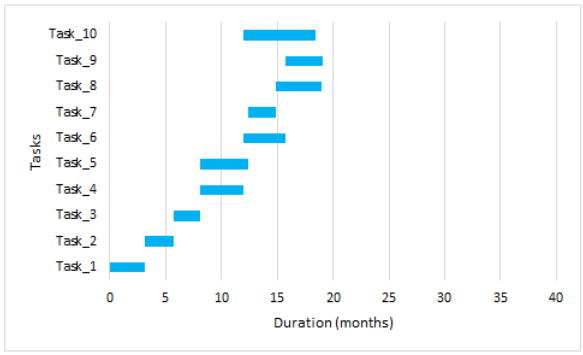



(c) 'T30_E15_SK4-5' instance

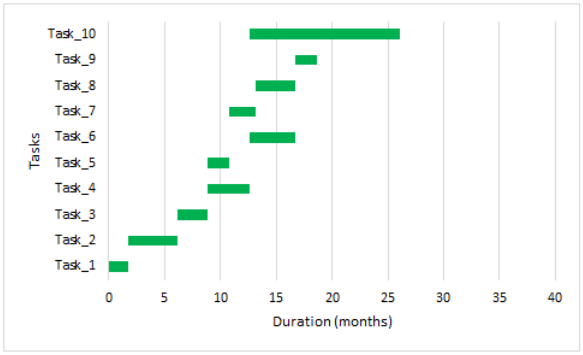
Figure 5.2: Results comparison between state-of-the-art algorithms. The final solutions of state-of-the-art algorithms are shown in the objective space $f_1$ and $f_2$.

the five peer algorithms show a poor performance in comparison to it. After BCE, OMOPSO is performing better by providing the minimized values of project duration and cost to the software project manager. SMPSO and Two_Arch2 are performing least well and do not provide good schedules. For the proposed model, BCE also shows, with statistical significance better performance against the other five state-of-the-art algorithms.

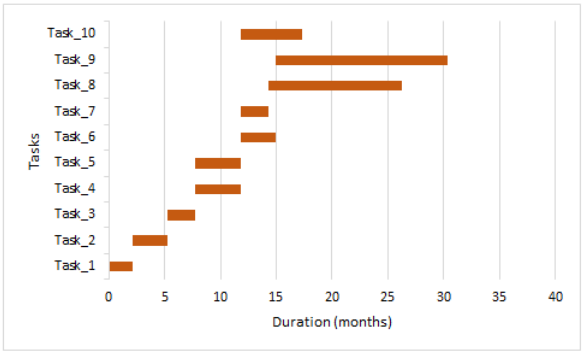
In Figure 5.3, schedules have been drawn from the results generated by each algorithm. We have selected the first run's results for data instance 'T10_E5_SK4-5'. The



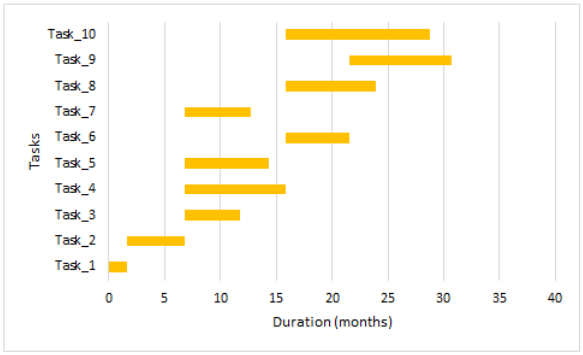

(a) BCE generated Schedule



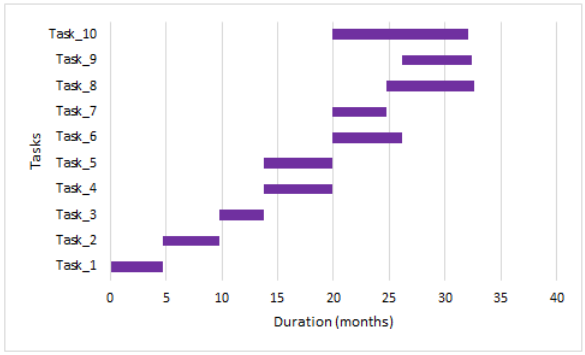

(b) NSGA-II generated Schedule



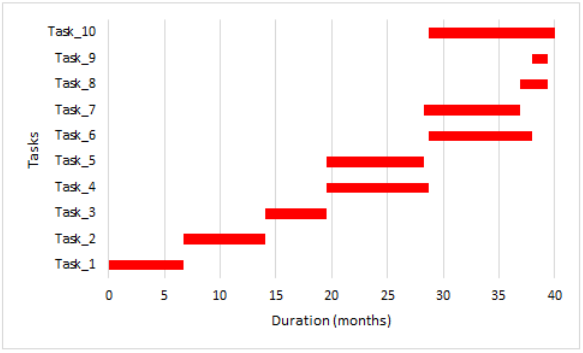

(c) Two_Arch2 generated Schedule




(d) NSGA-III generated Schedule




(c) OMOPSO generated Schedule




(d)SMPSO generated Schedule

Figure 5.3: Comparison of software project schedules based on EMO algorithms for 'T10_E5_SK4-5' data instance.

software project schedule generated from the BCE algorithm result while considering evolving employee's experience, has the shortest project duration as desired. Another interesting observation is that after BCE, NSGA-II performs better than all other algo-

rithms when drawing schedule. Two_Arch2 algorithm's results dominates NSGA-III in terms of project schedule. NSGA-III performs better than OMOPSO but worse than others. SMOPSO is performing least and returns the highest project duration.

To sum up, the distinct superiority of BCE is due to overcoming the following limitations of its peers: i) NSGA-III has been designed for more than three objectives and may not give very good results for two objective problems [51] as in our case, ii) OMOPSO loses the extrema of the Pareto front due to the use of $\varepsilon$-dominance [143] which results in losing optimal solutions, iii) SMPSO is among the slowest techniques in yielding a HVR greater than zero, that represents the bad exploration of search space resulting in limited solutions [117], iv) Two_Arch2 does not keep extreme points of the Pareto fronts which indicates loss of potential solutions [160], and v) NSGA-II lacks the uniform diversity among obtained non-dominated solutions while neglecting some potential areas in the search space [54]. We also observed that for data instances comprised of 42,43, and 91 tasks respectively, BCE performs better while other algorithms give trivial solutions.

## 5.6 Summary

In this chapter, a new model for the SPS problem is presented. Human resources are the main resources for the SPS problem. The software industries can save lots of money and time by appropriately assigning employees to tasks. The employee experience has emerged as a new factor significantly influencing the success of the software project. Hence, in this chapter, a new model for SPS problem is presented that takes into account employee experience. It's worth noting that employee experience along with learning ability over time (learning curve) has direct impact on reducing project duration and cost. The formulation of cost objective is also based on real-world scenarios.

To validate the proposed model, systematic experiments are carried out in two groups using benchmark and real-world problem instances. The first group of experiments provides extensive comparative studies between new and proposed models using a baseline algorithm (NSGA-II). The results reveal that the proposed model with the

inclusion of the employee experience factor reduces project duration and cost significantly. The second group of experiments analyses the performance of six existing state-of-the-art algorithms on the new model. The results show that BCE can achieve well converged and well distributed solutions sets with a good balance. Among six state-of-the-art EMO algorithms (BCE, NSGA-II, NSGA-III, Two_Arch2, OMOPSO, SMPSO), in general, BCE has distinct superiority among all by generating quality project schedules and reducing project cost and duration. These results respond to RQ5 (section 1.4) in Chapter 1.

# Chapter 6

# Conclusion

This thesis has introduced realistic models and search-based approaches to address the software project scheduling problem under a dynamic environment. This research gained inspiration from industrial practice and organizations' common issues that the project managers are more supported by providing insightful knowledge than exact solutions. In this direction, novel approaches are developed for multiple dynamic events which are very common during software project development. We also consider important human factors for the SPS problem. Hence, this research will help in extending algorithm contributions and problem solving for both industry and academia as mentioned in Table 6.1.

In this chapter, we present the contributions of this thesis. Section 6.1 summarized the research carried out. In doing so, the software project scheduling challenges stated at the beginning of this thesis are presented followed by future research directions in Section 6.2.

Table 6.1: Difficulties to which the proposed methods in this thesis provide solutions.

| Difficulties | Chapter 3 | Chapter 4 | Chapter 5 |
|---|---|---|---|
| Inefficiency of dealing with multiple project objectives (more than 2) | | √ | |
| Inefficiency of handling tasks' efforts uncertainties | | √ | |
| Lack of real-world data set | √ | √ | √ |
| Difficulty of dealing with dynamic events | √ | √ | |
| Inefficiency of dealing with important human aspects | | | √ |

## 6.1 Summary of Contributions

The major contributions of this thesis are summarised as under:

i. **Conceptual Model and MOEA-based Approach Towards 'Employee Turnover' Dynamic Event**. A model is presented based on real-world situation to handle the 'employee turnover' as a dynamic event which has not been considered before in the literature. This model supports both employee rescheduling and new employee recruitment scenarios. Based on this model, when turnover occurs, if rescheduling the remaining team can meet the manager's requirements then it is preferred first; else, we attempt to provide the manager with a set of options about what kind of person he/she could recruit within the budget. A new realistic way is also devised for the calculation of employee's proficiency for a task. Moreover, a multi-objective evolutionary approach is proposed to reschedule tasks to other employees when this dynamic event occurs. The newly proposed approach will also help software project managers in finding new employees to fulfil the missing skills when someone resigns in the middle of a software project. Additionally, this approach shows that how rescheduling of existing employees affects the project schedule. Furthermore, we also demonstrate that how our proposed approach helps the software project manager in recruiting a new employee to avoid any hiring delay. (Chapter 3)

ii. **A Heuristic-based Approach for 'New Employee Addition' Dynamic Event**. A novel multi-objective evolutionary algorithm, h_NEA, is proposed to deal with 'new employee addition' dynamic events. This has not been considered before in the literature. The h_NEA is based on a heuristic approach to deal with the SPS problem. Specifically, with an aim to minimize project cost and duration, heuristic strategies are designed for each project objective i.e. duration and cost. The duration heuristic will optimize the project duration by allocating a new employee to critical tasks identified by the critical path method, while the cost heuristic tries to minimize the project cost by employee replacement based on specific constraints. The employee replacement will only occur if there

is significant difference in cost. Further, it is also investigated that how the proposed heuristic affects other project objectives. (Chapter 4)

iii. **SPS Model based on Human Aspects**. A new SPS model based on an important human factor i.e. 'employee experience' is presented. This added parameter makes the SPS model more realistic. Based on this model, two project objectives are optimized, namely, project duration and cost. Moreover, the cost objective is formulated as in a real-world scenario. It is also demonstrated that how the proposed model significantly reduces project cost and duration as compared to state-of-the-art model. Furthermore, the six state-of-the-art algorithms (NSGA-II, NSGA-III, BCE, Two_Arch2, OMOPSO, SMPSO) performance is also evaluated on the proposed model, using 18 benchmark and six real-world data instances. The simulation results show that the BCE algorithm is particularly suitable for this new model as compared to peer algorithms - it can generate quality project plans which strike a good balance between project cost and duration. (Chapter 5)

## 6.2 Future work

This thesis introduces a number of new SBSE approaches and models for the software project scheduling problem to evaluate the advancements. To thoroughly explore their advantages, further investigation and more empirical studies are promised as indicated below.

**More factors and dynamic events inclusion.** We presented handling of employee related dynamic events ('employee turnover' and 'new employee addition') in Chapter 3 and Chapter 4 respectively, and inclusion of important human factor ('employee experience') into SPS model in Chapter 5. In the real-world, many other dynamic events can occur, e.g., software requirements cancellation and task precedence change. Consideration of other factors, e.g., task's slack time and group learning behaviour are also important which can be explored further. Group learning behaviour implies that if more than one employee is working on a task, they can learn from each other's

expertise which will also impact and reduce project duration.

Another aspect is consideration of software quality. The software quality can be treated at varying levels in the software project scheduling e.g. task, processes, phases, objective etc. Moreover, the quality can be measured with different models e.g. COCOMO [17]. In this thesis, cost, duration, robustness and stability objectives are considered with proposition of novel model and approaches so that these can be validated with the existing research based solutions. The software quality is the next step to be used in our validated model and approaches for knowledge contribution in the field of the SPS.

**Consideration of multiple critical paths.** In Chapter 4, we have presented a heuristic-based approach and optimized two objectives (i.e. duration and cost). Another area for future research is to explore the heuristic for other objectives as well. Also, h_NEA considers only one critical path, real-world software projects could have multiple critical paths. Other strategies could also be used to handle such situation. All of these enable us to have a deeper understanding of the near-optimal schedules in a dynamic environment.

**Scenario-based approach.** In our approach, we allocate employees to tasks based on optimized objectives. Applying the proposed multi-objective approaches in a variety of real-world scenarios would be highly desirable, both for further verifying the approaches and for solving complex real-life problems. For example, if two employees are not comfortable in working together then it would be best not to assign them the same tasks. Another example is that if a new urgent requirement is raised by the customer at the last minute, how to deal with that situation. Furthermore, we will possibly extend our approaches to recruit speculatively (i.e. recruit a staff member with skills that are most likely to be needed in future, even if there is no vacant staff role on any project).

**Multi-mode feature.** In Chapter 3 and Chapter 4, when a dynamic event occurs, a reschedule strategy is used and a new schedule is regenerated while considering stability objective. A further direction could be the introduction of multi-mode feature. This feature is based on dynamic events weights. If weight is very low, then match-up

strategies can be used which means partial adjustment in original schedule.

**Global Software Development.** In the recent software development industry, Global Software Development (GSD) is becoming a normal practice increasingly. This has been attributed to many factors, e.g., improved network infrastructure, learning and transfer of best practices, introduction of component-based architecture and increased time-to-market pressure [24]. Thus, more approaches could be designed to schedule software projects when development teams are based in different geographic locations.

**Incorporation of decision-maker preferences.** In our work, we select and implement one single solution (project schedule) from a set of solutions based on decision maker choice. The weight of each objective function in decision maker may be subjective. In future, we will consider the preferences supplied by the decision maker.

**Real-world application.** All the presented evolutionary multi-objective approaches, in this thesis were investigated mainly on benchmark and real-world data instances. These data sets have simulated data for the dynamic events. Once we get the real-world data with known dynamic events, further analyses should be performed. Additionally, we will test the developed approaches on larger project instances. Furthermore, multiple projects will also be considered. Another issue is that the schedules generated using these data sets cannot always match precisely with the project manager's preferences. This is called the resiliency of SPS solutions. It means to check if the solution generated by our approach is identical to software project manager choice. In the future, we will consult with the project manager to validate the software project schedules generated by our proposed approaches.

# Appendix A

# Survey of Software Engineering Process Models

This section performs critical evaluation of existing software engineering models as: waterfall, spiral, V-model, rational unified process (RUP) and agile models as a background information in Chapter 2.

**Waterfall Model**. This software process method is the most widely used and oldest model. It was introduced by Royce in 1970 [131]. The software development phases are modelled sequentially and it suggests that before starting the new phase, current phase must be completed and checked for any uncompleted tasks. Project managers expect that each task must be completed once it is started while using this model. But in reality, the case is different and for most of the software projects, project information and developer's knowledge becomes clearer with the passage of time. Software projects using waterfall requires the process to be restarted if some information is being missed at the start of the project. So, the waterfall model is suitable for the projects which have clear information available at the start of the project, project's objectives are defined in a clear manner, and probability of requirements change is very low.

**Spiral Model**. The Spiral model was introduced by Boehm in 1988 [18]. It is a risk-driven process model for medium to high risks projects. It is a combination of iterative

development model and linear sequential model. It overcomes major flaws of waterfall model and emphasis on risk analysis. There are four phases in spiral model: Planning, Risk Analysis, Engineering, and Evaluation. A software project repeatedly passes through these phases in an iterative manner. The project starts with the small set of requirements to be developed in initial iteration and guides the developers through the whole process. Experience is evolved in each iteration. Then, based on this experience, new additional requirements are added to the product in each of the following iterations. Software product develops in an incremental style. This iterative approach provides an adaptable environment which can deal with changing requirements. It also reduces risks as objectives are refined at the end of each iteration and risks are monitored. Spiral model is suitable to use when requirements are complex, significant changes are expected, cost and risk evaluation is important, and project is large and mission critical. It also provides strong documentation control. Unlike waterfall model, new requirements can be added at later stages of development without restarting the whole process. Another advantage is that software is produced in early iterations. On the other side, spiral model can be costly to use and it is not suitable for smaller projects. Project success is highly dependent on risk analysis phase and this phase requires highly specific expertise.

**V-Model**. The V-model means verification and validation model. It was first introduced for IT projects used for a federal department in Germany [144], and may be considered an extension of waterfall model. In V-model, each phase must be completed before the start of next phase. V-Shaped life cycle is a sequential path of execution of processes. By making association between analysis and development phases with the corresponding testing processes, it focuses on improving the communication between team members and customer. In this way both team members and client contribute to the project cooperatively. It is simple and easy to use and works well for small projects when requirements are clearly defined and unchangeable. It is proactive defect tracking model i.e. defects are found at early stages. It includes high risk in meeting customer expectations and very rigid and least flexible model. It should be chosen when many technical resources are available with needed technical expertise.

**Rational Unified Process**. Rational unified process (RUP) [97] is an object-oriented and web-enabled program development process developed by Rational software corporation, a division of IBM. It is an iterative software development methodology. In RUP, the software development process phases are named as inception, elaboration, construction, and transition. Each phase involves business modelling, analysis and design, implementation, testing, and deployment. Inception phase should be less than a week. Requirements are gathered and defined further in elaboration phase whereas some software development is also instantiated in this phase. Major software development starts in construction phase and testing is done in transition phase. Each phase may have one or more iterations and usually contains five work flows: requirements, analysis, design, implementation, and testing. Moreover, number of iterations in each phase depends on the project size.

RUP is a complete methodology with more emphasis on accurate documentation [96]. It has adaptive capability to deal with changing requirements. The time for integration of modules is very low as the integration process goes on throughout the software development life cycle. In RUP, components are also reusable so low development time is required.

**Agile Methods**. Agile is an umbrella term for a set of methods and practices based on values and principles expressed in Agile Manifesto 2001 [11]. Agile approaches are capable to respond to changes in order to succeed in an uncertain and turbulent environment. Agile methodology has replaced waterfall in most of the companies as a matter of choice. Agile is a time-boxed and iterative approach which builds the software in increments rather than delivering it all at once. Agile Manifesto is described following:

*i. Individuals and interactions over processes and tools*

*ii. Working software over comprehensive documentation*

*iii. Customer collaboration over contract negotiation*

*iv. Responding to change over following a plan*

Manifesto, instead of processes and tools, emphasis on people and the communication between them. It also includes that working software has priority over comprehensive documentation. In agile, all big tasks are broken into small tasks, and daily meetings ensure that work is on track and changes can be done even at the very end of the process. Several methodologies are being used in agile environment, e.g., scrum [133], extreme programming (XP) [10], feature driven development (FDD) [120], and more. Scrum and XP are most widely used agile methodologies. According to one survey, 66.7% of industries in Pakistan use scrum software process model.

# Bibliography


[1] Sami M Abbasi and Kenneth W Hollman. Turnover: The real bottom line. *Public personnel management*, 29(3):333–342, 2000. 60

[2] Tarek K Abdel-Hamid. A study of staff turnover, acquisition, and assimilation and their impact on software development cost and schedule. *Journal of Management Information Systems*, 6(1):21–40, 1989. 60, 65

[3] Sultan M Al Khatib and Joost Noppen. Benchmarking and comparison of software project human resource allocation optimization approaches. *ACM SIGSOFT Software Engineering Notes*, 41(6):1–6, 2017. 53

[4] Chowdhury Abdullah Al Mamun and Md Nazmul Hasan. Factors affecting employee turnover and sound retention strategies in business organization: a conceptual view. *Problems and Perspectives in Management*, (15, Iss. 1):63–71, 2017. 60

[5] Enrique Alba and J Francisco Chicano. Software project management with gas. *Information Sciences*, 177(11):2380–2401, 2007. 18, 19, 30, 48, 53, 54, 55, 56, 59, 74, 85, 95, 106, 113, 114, 115, 119

[6] Giuliano Antoniol, Massimiliano Di Penta, and Mark Harman. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In *10th International Symposium on Software Metrics, 2004. Proceedings.*, pages 172–183. IEEE, 2004. 51, 53

[7] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Theory of the hypervolume indicator: Optimal $\mu$-distributions and the choice of the reference point. In *Proceedings of the 10th ACM SIGEVO workshop on Foundations of Genetic Algorithms (FOGA)*, pages 87–102. -, 2009. 116

[8] Sebastian Baltes and Stephan Diehl. Towards a theory of software development expertise. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 187–200. ACM, 2018. 110, 114

[9] Ahilton Barreto, Márcio de O Barros, and Cláudia ML Werner. Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers & Operations Research*, 35(10):3073–3089, 2008. 53, 84

[10] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000. 133

[11] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001. 132

[12] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Robots and biological systems: towards a new bionics?*, pages 703–712. Springer, 1993. 40

[13] Flynn Bersin and Melian Mazor. *The employee experience: Culture, engagement, and beyond*, 2017. 106

[14] M Bichier and K-J Lin. Service-oriented computing. *Computer*, 39(3):99–101, 2006. 17

[15] Robert Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Professional, 2000. 17

[16] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003. 35

[17] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1981. 128

[18] Barry W Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988. 130

[19] Barry W. Boehm. Software risk management: principles and practices. *IEEE software*, 8(1):32–41, 1991. 50, 63, 72, 94

[20] Ricardo Britto, Pedro Santos Neto, Ricardo Rabelo, Werney Ayala, and Thiago Soares. A hybrid approach to solve the agile team allocation problem. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012. 53

[21] Frederick P. Brooks. *The Mythical Man-Month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995. 65, 85

[22] Jim Buchan, Stephen G MacDonell, and Jennifer Yang. Effective team onboarding in agile software development: techniques and goals. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE, 2019. 84

[23] He Can, Xing Jianchun, Zhu Ruide, Li Juelong, Yang Qiliang, and Xie Liqiang. A new model for software defect prediction using particle swarm optimization and support vector machine. In *2013 25th Chinese Control and Decision Conference (CCDC)*, pages 4106–4110. IEEE, 2013. 16, 112, 113

[24] Erran Carmel. *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR, 1999. 129

[25] Weng-Tat Chan, David KH Chua, and Govindan Kannan. Construction resource scheduling with genetic algorithms. *Journal of construction engineering and management*, 122(2):125–132, 1996. 48

[26] Shelvin Chand, Hemant Kumar Singh, and Tapabrata Ray. Finding robust solutions for resource constrained project scheduling problems involving uncertainties. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 225–232. IEEE, 2016. 51

[27] Carl K Chang, Chikuang Chao, Thinh T Nguyen, and Mark Christensen. Software project management net: a new methodology on software management. In *Computer Software and Applications Conference, 1998. COMPSAC'98. Proceedings. The Twenty-Second Annual International*, pages 534–539. IEEE, 1998. 18, 48

[28] Carl K. Chang, Mark J. Christensen, and Tao Zhang. Genetic algorithms for project management. *Annals of Software Engineering*, 11(1):107–139, Nov 2001. 18, 105

[29] Carl K Chang, Mark J Christensen, and Tao Zhang. Genetic algorithms for project management. *Annals of Software Engineering*, 11(1):107–139, 2001. 47, 48, 53

[30] Carl K Chang, Hsin-yi Jiang, Yu Di, Dan Zhu, and Yujia Ge. Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology*, 50(11):1142–1154, 2008. 18, 48, 53, 54, 73, 104

[31] Tao Chen, Miqing Li, and Xin Yao. On the effects of seeding strategies: a case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1419–1426, 2018. 17

[32] Tao Chen, Miqing Li, and Xin Yao. Standing on the shoulders of giants: Seeding search-based multi-objective optimization with prior knowledge for software service composition. *Information and Software Technology*, 114:155–175, 2019. 17

[33] Wei-Neng Chen and Jun Zhang. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, 39(1):1–17, 2013. 49, 53, 54, 59, 61, 107, 110, 112

[34] Francisco Chicano, Alejandro Cervantes, Francisco Luna, and Gustavo Recio. A novel multiobjective formulation of the robust software project scheduling problem. In *European Conference on the Applications of Evolutionary Computation*, pages 497–507. Springer, 2012. 49, 51, 53, 54

[35] Francisco Chicano, Francisco Luna, Antonio J Nebro, and Enrique Alba. Using multi-objective metaheuristics to solve the software project scheduling problem. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1915–1922. ACM, 2011. 53, 65, 77, 98, 112, 119

[36] Paul Clements and Linda Northrop. *Software product lines*. Addison-Wesley Boston, 2002. 17

[37] C. A. C. Coello and R. L. Becerra. Evolutionary multiobjective optimization in materials science and engineering. *Materials and Manufacturing Processes*, 24(2):119–129, 2009. 36

[38] C. A. C. Coello and G. B. Lamont. *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, 2004. 36

[39] C. A. Coello Coello and M. S. Lechuga. Mopso: a proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1051–1056 vol.2, May 2002. 47, 113

[40] CA Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE computational intelligence magazine*, 1(1):28–36, 2006. 49

[41] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007. 112

[42] Mike Cohn. *Succeeding with agile: software development using Scrum*. Pearson Education, 2010. 80

[43] Carlos Contreras-Bolton and Victor Parada. Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem. *PloS one*, 10(9), 2015. 47

[44] Broderick Crawford, Ricardo Soto, Franklin Johnson, Sanjay Misra, and Fernando Paredes. The use of metaheuristics to software project scheduling problem. In *International Conference on Computational Science and Its Applications*, pages 215–226. Springer, 2014. 53, 112, 119

[45] Broderick Crawford, Ricardo Soto, Franklin Johnson, and Eric Monfroy. Ants can schedule software projects. In *International Conference on Human-Computer Interaction*, pages 635–639. Springer, 2013. 53

[46] Broderick Crawford, Ricardo Soto, Franklin Johnson, Eric Monfroy, and Fernando Paredes. A max–min ant system algorithm to solve the software project scheduling problem. *Expert Systems with Applications*, 41(15):6634–6645, 2014. 53

[47] Charles Darwin and William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. Penguin Harmondsworth, 2009. 36

[48] Joaquim de Andrade, Leila Silva, André Britto, and Rodrigo Amaral. Solving the software project scheduling problem with hyper-heuristics. In Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz, and Jacek M. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 399–411, Cham, 2019. Springer International Publishing. 49

[49] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley, New York, 2001. 36

[50] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001. 15, 35, 112

[51] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601, 2014. 38, 113, 122

[52] Kalyanmoy Deb and Sachin Jain. Running performance metrics for evolutionary multi-objective optimization. 2002. 35

[53] Kalyanmoy Deb, Manikanth Mohan, and Shikhar Mishra. Evaluating the $\epsilon$-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions. *Evolutionary computation*, 13(4):501–525, 2005. 39, 53, 62, 67, 83, 89, 103

[54] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002. 37, 113, 117, 122

[55] Massimiliano Di Penta, Mark Harman, and Giuliano Antoniol. The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. *Software: Practice and Experience*, 41(5):495–519, 2011. 49

[56] Suzanne Dibble. *Keeping your valuable employees: retention strategies for your organization's most important resource*. John Wiley & Sons, 1999. 60

[57] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006. 42, 47

[58] Jim Duggan, Jason Byrne, and Gerard J Lyons. A task allocation optimizer for software construction. *IEEE software*, 21(3):76–82, 2004. 53

[59] Juan J Durillo and Antonio J Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011. 113

[60] Filomena Ferrucci, Mark Harman, Jian Ren, and Federica Sarro. Not going to take this anymore: multi-objective overtime planning for software engineering

projects. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 462–471. IEEE, 2013. 49

[61] Anthony Finkelstein and John Dowell. A comedy of errors: the london ambulance service case study. In *Proceedings of the 8th International Workshop on Software Specification and Design*, pages 2–4. IEEE, 1996. 107

[62] T. Friedrich, C. Horoba, and F. Neumann. Multiplicative approximations and the hypervolume indicator. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 571–578, 2009. 116

[63] T. Friedrich, F. Neumann, and C. Thyssen. Multiplicative approximations, optimal hypervolume distributions, and the choice of the reference point. *Evolutionary computation*, 23(1):131–159, 2015. 116

[64] M. Garza-Fabre, G. Toscano-Pulido, C. A. C. Coello, and E. Rodriguez-Tello. Effective ranking + speciation = Many-objective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2115–2122, 2011. 116

[65] Yujia Ge. Software project rescheduling with genetic algorithms. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 1, pages 439–443. IEEE, 2009. 51, 53

[66] Yujia Ge and Carl Chang. Capability-based project scheduling with genetic algorithms. In *2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06)*, pages 161–161. IEEE, 2006. 53

[67] Yujia Ge and Bin Xu. Dynamic staffing and rescheduling in software project management: A hybrid approach. *PloS one*, 11(6), 2016. 52

[68] Farhad Soleimanian Gharehchopogh, Isa Maleki, and Seyyed Reza Khaze. A novel particle swarm optimization approach for software effort estimation. *International Journal of Academic Research*, 6(2):69–76, 2014. 16, 112, 113

[69] Fred Glover. Tabu searchpart i. *ORSA Journal on computing*, 1(3):190–206, 1989. 15, 47

[70] Dimitri Golenko-Ginzburg and Aharon Gonik. Stochastic network project scheduling with non-consumable limited resources. *International Journal of Production Economics*, 48(1):29–37, 1997. 74, 84, 95, 106, 114

[71] Tad Gonsalves, Atsushi Ito, Ryo Kawabata, and Kiyoshi Itoh. Swarm intelligence in the optimization of software development project schedule. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 587–592. IEEE, 2008. 53

[72] Tad Gonsalves and Kiyoshi Itoh. Multi-objective optimization for software development projects. In *Lecture Notes in Engineering and Computer Science: International Multiconference of Engineers and Computer Scientist 2010*, pages 1–6, 2010. 53

[73] Stefan Gueorguiev, Mark Harman, and Giuliano Antoniol. Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1673–1680, 2009. 51, 53

[74] Wanjiang Han, Xiaoyan Zhang, Heyang Jiang, and Weijian Li. An ant colony optimization algorithm for software project management. In *2014 7th International Conference on Control and Automation*, pages 19–23. IEEE, 2014. 53

[75] Dinesh Bhagwan Hanchate. Analysis, mathematical modeling and algorithm for software project scheduling using bcga. In *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 3, pages 1–7. IEEE, 2010. 53

[76] Thomas Hanne and Stefan Nickel. A multiobjective evolutionary algorithm for scheduling and inspection planning in software development projects. *European Journal of Operational Research*, 167(3):663–678, 2005. 53

[77] Maciej Hapke, Andrzej Jaszkiewicz, and Roman Slowinski. Fuzzy project scheduling system for software development. *Fuzzy sets and systems*, 67(1):101–117, 1994. 50

[78] Mark Harman and Bryan F Jones. Search-based software engineering. *Information and software Technology*, 43(14):833–839, 2001. 14

[79] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Department of Computer Science, Kings College London, Tech. Rep. TR-09-03*, page 23, 2009. 112

[80] Tarek Hegazy and Kevin Petzold. Genetic optimization for dynamic project control. *Journal of construction engineering and management*, 129(4):396–404, 2003. 48

[81] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 517–528. IEEE, 2015. 17

[82] Alfredo G Hernández-Díaz, Luis V Santana-Quintero, Carlos A Coello Coello, and Julián Molina. Pareto-adaptive $\varepsilon$-dominance. *Evolutionary computation*, 15(4):493–517, 2007. 40

[83] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research*, 25(4):279–302, 1998. 107

[84] Robert M Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. Sip: Optimal product selection from feature models using many-objective evolution-

ary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(2):1–39, 2016. 17

[85] Khalil S. Hindi, Hongbo Yang, and Krzysztof Fleszar. An evolutionary algorithm for resource-constrained project scheduling. *IEEE Transactions on evolutionary computation*, 6(5):512–518, 2002. 49

[86] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. 15, 47

[87] Sun-Jen Huang and Nan-Hsing Chiu. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and software technology*, 48(11):1034–1045, 2006. 16, 112, 113

[88] Roger Ireland, Brian West, Norman Smith, and David I Shepherd. *Project management for it-related projects*. BCS, The Chartered Institute, 2012. 26

[89] H. Ishibuchi, Y. Hitotsuyanagi, N. Tsukamoto, and Y. Nojima. Many-objective test problems to visually examine the behavior of multiobjective evolution in a decision space. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 91–100, 2010. 116

[90] Ya-Hui Jia, Wei-Neng Chen, and Xiao-Min Hu. A pso approach for software project planning. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 7–8, 2014. 53, 54

[91] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995. 40

[92] Harold Kerzner. *Project management: a systems approach to planning, scheduling, and controlling*. John Wiley & Sons, 2017. 18

[93] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. 15, 47

[94] Peter Knauber, Jesus Bermejo, Günter Böckle, Julio Cesar Sampaio do Prado Leite, Frank van der Linden, Linda Northrop, Michael Stark, and David M Weiss. Quantifying product line benefits. In *International Workshop on Software Product-Family Engineering*, pages 155–163. Springer, 2001. 17

[95] Josiane Kroll, Shai Friboim, and Hadi Hemmati. An empirical study of search-based task scheduling in global software development. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 183–192. IEEE, 2017. 53

[96] Per Kroll and Philippe Kruchten. *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional, 2003. 132

[97] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004. 132

[98] Divya Kumar and KK Mishra. The impacts of test automation on software's cost, quality and time to market. *Procedia Computer Science*, 79:8–15, 2016. 16

[99] Kenneth C Laudon and Jane P Laudon. *Management information systems*. Prentice Hall PTR, 1999. 65

[100] Sanja Lazarova-Molnar and Rabeb Mizouni. A simulation-based approach to enhancing project schedules by the inclusion of remedial action scenarios. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 761–772. IEEE, 2011. 51

[101] Huaizhong Li and Chiou Peng Lam. Software test data generation using ant colony optimization. In *International conference on computational intelligence*, pages 1–4, 2004. 17

[102] Huaizhong Li and Chiou Peng Lam. An ant colony optimization approach to test sequence generation for state based software testing. In *Fifth International Conference on Quality Software (QSIC'05)*, pages 255–262. IEEE, 2005. 17, 112, 113

[103] Miqing Li, Shengxiang Yang, and Xiaohui Liu. A test problem for visual investigation of high-dimensional multi-objective search. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2140–2147. IEEE, 2014. 40

[104] Miqing Li, Shengxiang Yang, and Xiaohui Liu. Pareto or non-pareto: Bi-criterion evolution in multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 20(5):645–665, 2016. 38, 113

[105] Miqing Li, Shengxiang Yang, Xiaohui Liu, and Ruimin Shen. A comparative study on evolutionary algorithms for many-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 261–275. Springer, 2013. 40

[106] Miqing Li and Xin Yao. Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Computing Surveys (CSUR)*, 52(2):26, 2019. 44

[107] Zhizhong Li and Günther Ruhe. Uncertainty handling in tabular-based requirements using rough sets. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 678–687. Springer, 2005. 50

[108] Francisco Luna, Francisco Chicano, and Enrique Alba. Robust solutions for the software project scheduling problem: a preliminary analysis. *International Journal of Metaheuristics*, 2(1):56–79, 2012. 53, 112, 119

[109] Francisco Luna, David L González-Álvarez, Francisco Chicano, and Miguel A Vega-Rodríguez. On the scalability of multi-objective metaheuristics for the software scheduling problem. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 1110–1115. IEEE, 2011. 53

[110] Francisco Luna, David L González-Álvarez, Francisco Chicano, and Miguel A Vega-Rodríguez. The software project scheduling problem: A scalability analysis of multi-objective metaheuristics. *Applied Soft Computing*, 15:136–148, 2014. 49, 53, 112, 119

[111] Leandro L Minku, Dirk Sudholt, and Xin Yao. Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. *IEEE Transactions on Software Engineering*, 40(1):83–102, 2014. 18, 49, 53, 107

[112] Leandro L Minku and Xin Yao. An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation. In *Proceedings of the 9th international conference on predictive models in software engineering*, pages 1–10, 2013. 16, 112, 113

[113] Audris Mockus and James D Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 503–512. IEEE, 2002. 110

[114] Jo Ellen Moore and Lisa A Burke. How to turn aroundturnover culture'in it. *Communications of the ACM*, 45(2):73–78, 2002. 60

[115] J. Morgan and M. Goldsmith. *The Employee Experience Advantage: How to Win the War for Talent by Giving Employees the Workspaces they Want, the Tools they Need, and a Culture They Can Celebrate*. Wiley, 2017. 106, 107

[116] N. Nan and D. E. Harter. Impact of budget and schedule pressure on software development cycle time and effort. *IEEE Transactions on Software Engineering*, 35(5):624–637, Sep. 2009. 50, 82, 106

[117] Antonio J Nebro, Juan José Durillo, Jose Garcia-Nieto, CA Coello Coello, Francisco Luna, and Enrique Alba. Smpso: A new pso-based metaheuristic for multi-objective optimization. In *2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*, pages 66–73. IEEE, 2009. 42, 49, 113, 122

[118] Natasha Nigar. Model-based dynamic software project scheduling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 1042–1045. ACM, 2017. 18, 107

[119] Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, 12(4):417, 2009. 75, 97

[120] Steve R Palmer and Mac Felsing. *A practical guide to feature-driven development*. Pearson Education, 2001. 133

[121] H Van Dyke Parunak. Characterizing the manufacturing scheduling problem. *Journal of manufacturing systems*, 10(3):241–259, 1991. 19

[122] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007. 20

[123] A. Ponsich, A. L. Jaimes, and C. A. C. Coello. A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications. *IEEE Transactions on Evolutionary Computation*, 17(3):321–344, 2013. 36

[124] Roger S Pressman. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005. 50

[125] James L Price. *The study of turnover*. Iowa State Press, 1977. 60

[126] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy MA Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10):1605–1614, 2007. 83, 89

[127] Jian Ren, Mark Harman, and Massimiliano Di Penta. Cooperative co-evolutionary optimization of software project staff assignments and job scheduling. In *International Symposium on Search Based Software Engineering*, pages 127–141. Springer, 2011. 49, 53

[128] Allan Vinicius Rezende, Leila Silva, André Britto, and Rodrigo Amaral. Software project scheduling problem in the context of search-based software engineering: A systematic review. *Journal of Systems and Software*, 155:43–56, 2019. 50

[129] Daniel Rodríguez, Mercedes Ruiz, José C Riquelme, and Rachel Harrison. Multi-objective simulation optimisation in software project management. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1883–1890, 2011. 53, 54

[130] Jiang Rong, Liao Hongzhi, Yu Jiankun, Feng Tao, Zhao Chenggui, and Li Junlin. A model based on information entropy to measure developer turnover risk on software project. In *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pages 419–422. IEEE, 2009. 62

[131] Winston W Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338, 1987. 130

[132] Thomas L Saaty and Luis G Vargas. Comparison of eigenvalue, logarithmic least squares and least squares methods in estimating ratios. *Mathematical modelling*, 5(5):309–324, 1984. 45

[133] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002. 133

[134] Randall F Scott and Dick B Simmons. Predicting programming group productivity a communications model. *IEEE Transactions on Software Engineering*, (4):411–414, 1975. 65

[135] Dongwon Seo, Donghwan Shin, and Doo-Hwan Bae. Quality based software project staffing and scheduling with cost bound. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 269–276. IEEE, 2015. 53

[136] Xiaohong Shan, Guorui Jiang, and Tiyun Huang. The optimization research on the human resource allocation planning in software projects. In *2010 International Conference on Management and Service Science*, pages 1–4. IEEE, 2010. 53

[137] Xiao-Ning Shen, Leandro L Minku, Naresh Marturi, Yi-Nan Guo, and Ying Han. A q-learning-based memetic algorithm for multi-objective dynamic software project scheduling. *Information Sciences*, 428:1–29, 2018. 18, 52, 53, 61, 92, 107, 109

[138] Xiao-Ning Shen and Xin Yao. Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems. *Information Sciences*, 298:198–224, 2015. ix, x, xi, 44, 45, 77, 98, 100, 101, 117, 119

[139] Xiaoning Shen, Leandro L Minku, Rami Bahsoon, and Xin Yao. Dynamic software project scheduling through a proactive-rescheduling method. *IEEE Transactions on Software Engineering*, 42(7):658–686, 2016. 26, 52, 53, 54, 58, 59, 61, 71, 72, 75, 76, 84, 89, 93, 95, 96, 97, 98, 99, 100, 101, 102, 103, 105, 107, 108, 109, 116, 117

[140] Martin L Shooman. *Software Engineering: Design, Reliability, and Management*. McGraw-Hill, 1980. 65

[141] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. Measuring and modeling programming experience. *Empirical Software Engineering*, 19(5):1299–1334, 2014. 110

[142] Margarita Reyes Sierra and Carlos A Coello Coello. Improving pso-based multi-objective optimization using crowding, mutation and $\varepsilon$-dominance. In *International conference on evolutionary multi-criterion optimization*, pages 505–519. Springer, 2005. 42, 113

[143] Margarita Reyes Sierra and Carlos A Coello Coello. Improving pso-based multi-objective optimization using crowding, mutation and $\varepsilon$-dominance. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 505–519. Springer, 2005. 122

[144] I Somerville. *Software Engineering*. Addison-Wesley, 1992. 131

[145] Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010. 82, 87

[146] Online Source. *Know Your Worth*, 2019. 107, 111

[147] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994. 37

[148] Praveen Ranjan Srivastava and Km Baby. Automated software testing using metahurestic technique based on an ant colony optimization. In *2010 international symposium on electronic system design*, pages 235–240. IEEE, 2010. 17, 112, 113

[149] Constantinos Stylianou and Andreas S Andreou. Intelligent software project scheduling and team staffing with genetic algorithms. In *Artificial Intelligence Applications and Innovations*, pages 169–178. Springer, 2011. 53

[150] Constantinos Stylianou, Simos Gerasimou, and Andreas S Andreou. A novel prototype tool for intelligent software project scheduling and staffing enhanced with personality factors. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, volume 1, pages 277–284. IEEE, 2012. 53

[151] V Suresh and Dipak Chaudhuri. Dynamic scheduling: a survey of research. *International journal of production economics*, 32(1):53–63, 1993. 19

[152] Bharti Suri and Pooja Jajoria. Using ant colony optimization in software development project scheduling. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2101–2106. IEEE, 2013. 53

[153] Madjid Tavana, Amir-Reza Abtahi, and Kaveh Khalili-Damghani. A new multi-objective multi-mode model for solving preemptive time–cost–quality trade-off project scheduling problems. *Expert Systems with Applications*, 41(4):1830–1846, 2014. 49

[154] Vicente Valls, Francisco Ballestin, and Sacramento Quintanilla. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 185(2):495–508, 2008. 49

[155] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm test suites. In *SAC*, volume 99, pages 351–357. Citeseer, 1999. 44, 117

[156] Miguel Ángel Vega-Velázquez, Abel García-Nájera, and Humberto Cervantes. A survey on the software project scheduling problem. *International Journal of Production Economics*, 202:145–161, 2018. 50

[157] T. Wagner, N. Beume, and B. Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *Evolutionary Multi-Criterion Optimization (EMO)*, pages 742–756. -, 2007. 116

[158] Romi Satria Wahono and Nanna Suryana. Combining particle swarm optimization based feature selection and bagging technique for software defect prediction. *International Journal of Software Engineering and Its Applications*, 7(5):153–166, 2013. 16, 112, 113

[159] Romi Satria Wahono, Nanna Suryana, and Sabrina Ahmad. Metaheuristic optimization based feature selection for software defect prediction. *Journal of Software*, 9(5):1324–1333, 2014. 16, 112, 113

[160] Handing Wang, Licheng Jiao, and Xin Yao. Two_arch2: An improved two-archive algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 19(4):524–541, 2015. 39, 113, 122

[161] J.D. Wiest and F.K. Levy. *A management guide to PERT/CPM: with GERT/PDM/DCPM and other networks*. Prentice-Hall, 1977. 84, 85, 106

[162] Andreas Windisch, Stefan Wappler, and Joachim Wegener. Applying particle swarm optimization to software testing. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1121–1128, 2007. 17

[163] A Wise. Little-jil 1.5 language report. department of computer science, university of massachusetts. Technical report, Amherst UM-CS-2006-51, 2006. 51

[164] Muh-Cherng Wu and Shih-Hsiung Sun. A project scheduling and staff assignment model considering learning effect. *The International Journal of Advanced Manufacturing Technology*, 28(11-12):1190–1195, 2006. 107

[165] Xiuli Wu, Pietro Consoli, Leandro Minku, Gabriela Ochoa, and Xin Yao. An evolutionary hyper-heuristic for the software project scheduling problem. In *International Conference on Parallel Problem Solving from Nature*, pages 37–47. Springer, 2016. 49

[166] Yi Xiang, Yuren Zhou, Zibin Zheng, and Miqing Li. Configuring software product lines by combining many-objective optimization and sat solvers. *ACM Trans. Softw. Eng. Methodol.*, 26(4), February 2018. 17

[167] Jing Xiao, Xian-Ting Ao, and Yong Tang. Solving software project scheduling problems with ant colony optimization. *Computers & Operations Research*, 40(1):33–46, 2013. 49, 53

[168] Jing Xiao, Mei-Ling Gao, and Min-Mei Huang. Empirical study of multi-objective ant colony optimization to software project scheduling problems. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 759–766, 2015. 53

[169] Junchao Xiao, Leon J Osterweil, Qing Wang, and Mingshu Li. Dynamic resource scheduling in disruption-prone software development environments. In *International Conference on Fundamental Approaches to Software Engineering*, pages 107–122. Springer, 2010. 51, 53

[170] Junchao Xiao, Qing Wang, Mingshu Li, Qiusong Yang, Lizi Xie, and Dapeng Liu. Value-based multiple software projects scheduling with genetic algorithm. In *International Conference on Software Process*, pages 50–62. Springer, 2009. 48, 53, 54

[171] Shengxiang Yang, Miqing Li, Xiaohui Liu, and Jinhua Zheng. A grid-based evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 17(5):721–736, 2013. 40

[172] Virginia Yannibelli and Analía Amandi. A knowledge-based evolutionary assistant to software development project scheduling. *Expert Systems with Applications*, 38(7):8403–8413, 2011. 53, 54

[173] Janez Zerovnik. Heuristics for np-hard optimization problems - simpler is better!? *Logistics and Sustainable Transport*, 6:1–10, 12 2015. 85

[174] G. Zhang, Z. Su, M. Li, F. Yue, J. Jiang, and X. Yao. Constraint handling in nsga-ii for solving optimal testing resource allocation problems. *IEEE Transactions on Reliability*, 66(4):1193–1212, 2017. 17, 112, 113

[175] Wei Zhang, Yun Yang, Junchao Xiao, Xiao Liu, and Muhammad Ali Babar. Ant colony algorithm based scheduling for handling software project delay. In *Proceedings of the 2015 International Conference on Software and System Process*, pages 52–56, 2015. 53, 104

[176] Wei Zheng, Robert M Hierons, Miqing Li, XiaoHui Liu, and Veronica Vinciotti. Multi-objective optimisation for regression testing. *Information Sciences*, 334:1–16, 2016. 17, 112, 113

[177] A. Zhou, B. Y. Qu, H. Li, S. Z. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011. 36

[178] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Quality assessment of pareto set approximations. In *Multiobjective Optimization*, pages 373–404. Springer, 2008. 99, 120

[179] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999. 44, 116