University of Cyprus

OPTIMIZING RESOURCE ALLOCATION AND TASK SCHEDULING IN SOFTWARE DEVELOPMENT

Constantinos Stylianou

A Dissertation
Submitted to the University of Cyprus
in Partial Fulfilment of the
Requirements for the Degree of
Doctor of Philosophy

Recommended for Acceptance
by the Department of Computer Science
May, 2016

2016

APPROVAL PAGE

OPTIMIZING RESOURCE ALLOCATION AND TASK SCHEDULING IN SOFTWARE DEVELOPMENT

Constantinos Stylianou

The present doctoral dissertation was submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at the Department of Computer Science of the University of Cyprus and was approved on 27th May 2016 by the members of the Examination Committee.

University of Cyprus

May, 2016

DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at the Department of Computer Science of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Constantinos Stylianou

ΠΕΡΙΛΗΨΗ

Η διαχείριση έργων λογισμικού αποτελείται από διάφορες δραστηριότητες προγραμματισμού, οργάνωσης, στελέχωσης, καθοδήγησης και ελέγχου. Οι αποφάσεις που παίρνουν οι διαχειριστές έργων λογισμικού σ' αυτές τις δραστηριότητες, καθώς και οι διάφορες πρακτικές που ακολουθούνται, πιθανόν να επηρεάσουν την επιτυχία ενός έργου λογισμικού. Η έρευνα που παρουσιάζεται σ' αυτή τη διδακτορική διατριβή επικεντρώνεται ειδικά στην περιοχή του προγραμματισμού έργων και, συγκεκριμένα, στις δραστηριότητες ανάθεσης πόρων και χρονοπρογραμματισμού εργασιών, μέσα στις οποίες ένας διαχειριστής έργων πρέπει να αποφασίσει ποιος θα κάνει τι και πότε μέσα σ' ένα έργο λογισμικού.

Σ' αυτές τις δραστηριότητες, οι διαχειριστές έργων χρειάζονται, συνήθως ταυτόχρονα, να αναθέσουν μηχανικούς λογισμικού σε εργασίες και να προγραμματίσουν τον χρόνο εκτέλεσης των εργασιών με σκοπό την ικανοποίηση διαφόρων στόχων και προϋποθέσεων. Ωστόσο, η ανάληψη αυτών των δραστηριοτήτων είναι συνήθως πρόκληση για τους διαχειριστές έργων λογισμικού επειδή συνοδεύονται από αντικρουόμενους περιορισμούς χρόνου, κόστους και ποιότητας, οι οποίοι δύσκολα εξισορροπούνται αποτελεσματικά. Επιπλέον, επειδή οι ανθρώπινοι πόροι θεωρούνται οι μοναδικοί διαθέσιμοι πόροι για μια εταιρεία παραγωγής λογισμικού, είναι σημαντικό οι πληροφορίες που χρησιμοποιούνται σ' αυτές τις δραστηριότητες να περιλαμβάνουν και τα χαρακτηριστικά των εργασιών που θα εκτελεστούν, αλλά και τα χαρακτηριστικά των πόρων που θα εκτελέσουν αυτές τις εργασίες.

Μια νέα τάση στην περιοχή αφορά στην συμπερίληψη της προσωπικότητας των μηχανικών λογισμικού. Διάφορες μελέτες έχουν παρατηρήσει την επίδραση των τύπων προσωπικότητας πάνω σε πτυχές όπως την απόδοση και την επαγγελματική ικανοποίηση, οι οποίες πιθανόν να μπορούν να συνεισφέρουν στην επιτυχία ενός έργου λογισμικού. Επίσης, έχουν γίνει προσπάθειες καθορισμού των επιθυμητών τύπων προσωπικότητας που απαιτούν τα διάφορα επαγγέλματα παραγωγής λογισμικού, προκειμένου να ανατίθενται εργασίες σε μηχανικούς λογισμικού που ταιριάζουν καλύτερα στην προσωπικότητά τους.

Η διατριβή παρέχει μια λεπτομερή περιγραφή διαφόρων ερευνητικών προσπαθειών που πραγματοποιήθηκαν υιοθετώντας μεθόδους πολυστοχικής βελτιστοποίησης με σκοπό την επίλυση του

προβλήματος της ανάθεσης πόρων και χρονοπρογραμματισμού εργασιών στα έργα παραγωγής λογισμικού. Οι προτεινόμενες προσεγγίσεις που περιγράφονται σ' αυτές τις προσπάθειες χρησιμοποιούν πρακτικά κριτήρια που σχετίζονται με την παραγωγή λογισμικού, καθώς και αυστηρές, ρεαλιστικές προϋποθέσεις. Επικεντρώνονται κυρίως στον χειρισμό της μη-εναλλάξιμης φύσης των ανθρώπινων πόρων, συμπεριλαμβάνοντας παράγοντες όπως την προσπάθεια και τις δεξιότητες που απαιτούν οι εργασίες, τα επίπεδα εμπειρίας και τον ρυθμό παραγωγικότητας των μηχανικών λογισμικού, και επιπρόσθετα τον τρόπο με τον οποίο οι μηχανικοί λογισμικού εργάζονται μαζί ανάλογα με τον τύπο της εργασίας που εκτελούν. Επιπλέον, μια από τις προτεινόμενες προσεγγίσεις επιχειρεί να αναθέσει πόρους σε εργασίες βάσει την καταλληλότητα του τύπου προσωπικότητας των μηχανικών λογισμικού.

Τα αποτελέσματα των διαφόρων πειραμάτων που πραγματοποιήθηκαν για την αξιολόγηση των προσεγγίσεων δείχνουν ότι οι μέθοδοι βελτιστοποίησης μπορούν πράγματι να χειριστούν επαρκώς τους συγκεκριμένους στόχους και περιορισμούς που υιοθετήθηκαν, και ότι οι προτεινόμενες προσεγγίσεις έχουν τη δυνατότητα να αποτελέσουν μια πιο αποτελεσματική και πρακτική μέθοδο για την ανάθεση πόρων και χρονοπρογραμματισμό εργασιών σε έργα παραγωγής λογισμικού.

ABSTRACT

Software project management consists of a number of planning, organizing, staffing, directing and controlling activities. Decisions taken by software project managers in these activities, as well as the different practices followed, are likely to influence the success of a software project. The research presented in this doctoral dissertation focuses specifically on the area of project planning and, in particular, on the activities of resource allocation and task scheduling, in which project managers must decide who will do what and when in a software project.

In these activities, project managers are required to assign developers to tasks and plan the execution of tasks, often simultaneously, with the aim of satisfying several goals and assumptions. However, these activities are often challenging to undertake because they are accompanied by conflicting time, budget and quality constraints, which project managers find difficult to balance effectively. Furthermore, because human resources are considered the only type of resource available for software development companies, it is important that the information used for these activities consists of both the characteristics of the tasks to be carried out, as well as the attributes of the resources that will carry out these tasks.

A leading trend in the area involves taking into account the personality of developers. A number of studies have observed the effects of personality types on aspects such as performance and job satisfaction, which can potentially contribute towards the success of a project. Also, there have been attempts to determine the personality type required for different software development professions in order to allocate developers to tasks that better suit their personality.

The dissertation provides a detailed account of several research attempts carried out that adopt multiobjective optimization methods in order to solve the problem of resource allocation and task scheduling in software development. The proposed approaches described in these attempts use practical software-related criteria, as well as strict, realistic assumptions. They mainly focus on dealing with the noninterchangeable nature of human resources by including factors such as the effort and skills required by tasks, the experience levels and productivity rates of software developers, in addition to the way developers work together depending on the type of task carried out. Furthermore, one of the proposed approaches attempts to allocate resources based on the suitability of the personality type of developers.

The results of various experiments carried out to evaluate the approaches show that the specific objectives and constraints adopted can indeed be handled adequately by the optimization methods, and that the proposed approaches have the potential to constitute a more effective and practical method for resource allocation and task scheduling in software development.

ACKNOWLEDGEMENTS

With the completion of my Ph.D. dissertation and, in general, the conclusion of my doctoral studies at the Department of Computer Science of the University of Cyprus, I would like to express my sincere gratitude to all those who helped me achieve my goals by accompanying me on this long, difficult, but definitely rewarding, journey.

To begin with, I would like to deeply thank my research supervisor, Associate Professor Dr Andreas Andreou, for providing me with his invaluable help, guidance and backing throughout our years of working together. He was a fountainhead of immense knowledge and motivation by always forcing me to challenge myself and my ideas, while all the time providing critical feedback and direction. I genuinely could not have imagined having a better adviser and mentor for my Ph.D. studies. I would also like to thank Professor Dr Constantinos Pattichis, who undertook the co-supervision of my studies following the departure of Dr Andreas Andreou from the University of Cyprus. He was always ready and willing to give assistance whenever I needed his expert opinion and advice. I truly respect the confidence he has shown in me over the years.

Besides my advisers, I would like to thank the rest of my examining committee: Associate Professor Dr Lefteris Angelis, Associate Professor Dr Andreas Nearchou, Assistant Professor Dr George Pallis and Assistant Professor Dr Georgia Kapitsaki, for their insightful comments and encouragement, but also for the hard question that incentivized me to widen my research from various perspectives.

Furthermore, I would like to express my sincere appreciation to the members, both past and present, of the Software Engineering and Intelligent Information Systems Research Laboratory. I am tremendously grateful to them for sharing their constructive criticism and friendly advice, both on a professional and personal level.

Also, I would like to thank all my close friends, who offered their unlimited support and encouragement throughout my studies. They were a constant boost for my resilience and persistence. For that, I will forever be grateful.

Last, but never least, I would like to thank my family: my mother and father, my sister, my brother-in-law, my nephews and my niece. They were an inexhaustible source of inspiration, patience and perseverance throughout my efforts to complete my studies. I am eternally grateful for

their tremendous belief in me and the unconditional support they provided that allowed me to pursue my goal of obtaining a doctoral degree. They were the ones that kept me focused and believing that everything works out in the end.

CREDITS

[1] C. Stylianou and A. S. Andreou, "Intelligent software project scheduling and team staffing with genetic algorithms," in *Artificial Intelligence Applications and Innovations: Proceedings of the 7th IFIP International Conference on Artificial Intelligence Applications and Innovations, Volume 2* (L. S. Iliadis, I. Maglogiannis and H. Papadopoulos, eds.), vol. 364 of *IFIP Advances in Information and Communication Technology*, pp. 169–178, Berlin, Germany: Springer-Verlag, Sep. 2011.

[2] S. Gerasimou, C. Stylianou and A. S. Andreou, "An investigation of optimal project scheduling and team staffing in software development using particle swarm optimization," in *Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 1*, pp. 168–171, SciTePress, Jul. 2012.

[3] C. Stylianou, S. Gerasimou and A. S. Andreou, "A multi-objective genetic algorithm for software development team staffing based on personality types," in *Artificial Intelligence Applications and Innovations: Proceedings of the 8th IFIP International Conference on Artificial Intelligence Applications and Innovations, Volume 1* (L. S. Iliadis, I. Maglogiannis and H. Papadopoulos, eds.), vol. 381 of *IFIP Advances in Information and Communication Technology*, pp. 37–47, Berlin, Germany: Springer-Verlag, Sep. 2012.

[4] C. Stylianou and A. S. Andreou, "A novel prototype tool for intelligent software project scheduling and staffing enhanced with personality factors," in *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence*, pp. 277–287, IEEE Computer Society, Nov. 2012.

[5] C. Stylianou and A. S. Andreou, "A multi-objective genetic algorithm for intelligent software project scheduling and team staffing," *Intelligent Decision Technologies*, vol. 7, no. 1, pp. 59–80, Jan. 2013.

[6] C. Stylianou and A. S. Andreou, *Human resource allocation and scheduling for software project management*, ch. 4, pp. 73–106. Software Project Management in a Changing World, Berlin, Germany: Springer-Verlag, 2014.

[7] C. Stylianou and A. S. Andreou, "Investigating the impact of developer productivity, task interdependence type and communication overhead in a multi-objective optimization approach for software project planning," *Advances in Engineering Software*, vol. 98, pp. 79–96, Aug. 2016.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

One of the biggest problems still facing the software development industry is the high rate of challenged and failed software projects. According to the 2015 Standish Group CHAOS Report [1]:

- 29% of software projects were delivered on time, within budget and with the required functionality,
- 52% of software projects were delivered late, over budget and/or with less than the required functionality, and
- 19% of software projects were cancelled prior to completion or delivered but never used.

These figures suggest that only three out of ten projects are likely to be completed successfully. The group also reveals that small projects have a greater chance of succeeding than larger projects. In fact, around 92% of successful projects are of small-to-medium sized, whereas the remaining 8% are classed as large or grand. Furthermore, projects following a traditional waterfall life-cycle model have a much lower success rate (11%) compared to projects developed with agile processes (39%). Considering the ever-increasing size and complexity of modern-day software products, development companies are facing escalating pressure to deliver products to customers sooner, cheaper and with higher quality than their competitors in order to remain viable in today's competitive software market. This pressure, however, hinders the probability of software projects success.

Tackling the issue of low software project success rates has been the focal point of many researchers in the field of software engineering for a number of years. A variety of reasons have

been suggested as the cause of this problem, both from a technical perspective (for example, the ambiguous and unstable nature of requirements, the lack of end-user involvement, contradicting stakeholder policies and the poor design of system architectures) and from a project management perspective (for example, the lack of support from senior executives, competing priorities, cultural or ethical misalignment and the lack of risk management). One of the most significant causes is linked to the lack of proper practices to help project managers in their planning, organizing, staffing, directing and control activities. Inadequate or insufficient practices can cause software project managers to make incorrect decisions within these activities; for example, employing inaccurate cost estimation techniques that lead to budget and schedule explosions or using inadequate progress reporting and tracking tools that creates problems in change requests. Providing project managers with effective methods, tools and techniques for these practices can, therefore, be considered central to the success of a software project.

1.1 Problem statement

Many researchers consider that practices related to planning activities are the most crucial for the success of a software project, especially because these activities are required at the start of a project; any problems, inaccuracies or oversights are likely to be propagated as the project progresses. Specifically, planning activities require software project managers to undertake various budgeting and scheduling tasks in order to determine how the software product will be built, how much it will cost and how long it will take to deliver it to the customer. In order to answer these questions, two activities need to be carried out, namely resource allocation and task scheduling, during which software project managers decide what needs to be done, when and by whom. These two activities are considered extremely important for the success of a software project because, on the one hand, inaccurate task scheduling may cause significant delays in delivery and budget overruns and, on the other hand, improper resource allocation can lead to an undesired low level of quality in the software products.

Planning practices adopted by software project companies should aim to assign tasks to developers and arrange the execution of tasks simultaneously in a way that satisfies the objectives determined by a project manager. However, despite the fact that many research attempts have proposed approaches to solve this problem, there is still a lack of adequate tools and methods to help software project managers carry out resource allocation and task scheduling for their software

projects. Several of the approaches proposed up until now have significant drawbacks, particularly because they often fail to take into account specific development-related factors and/or tend to relax certain assumptions, both of which severely limit their applicability in actual real-world projects in the software development industry.

Human resources (that is, software developers) feature prominently in the resource allocation and tasks scheduling activities of software project managers. As a matter of fact, developers are often regarded as 'human capital' and considered the only type of resource available in software development companies [2]. As a consequence, human factors can greatly influence not only decisions within planning activities, but also decisions in other activities (such as staffing and directing). Researchers therefore consider these factors critical for the success of software projects since they can affect aspects such as productivity, performance and job satisfaction, as well as software quality. For this reason, a primary concern for software project managers is to make sure that the resource allocation and task scheduling activities take into account the human-centred factors of software developers and not just the technical characteristics of the software project.

One interesting aspect of human resources involves the personality of software developers. Researchers have conducted several studies examining the importance of personality in software development and how this aspect can help in allocating resources or forming development teams. For example, there have been a number of studies that attempt to profile the personality type required by various software development professions [3]. These profiles can then be used to help software project managers allocate resources more effectively by assigning developers to tasks whose personality better suits the personality profile of the task. Other studies concentrate on assessing which combination of personalities yield better-performing teams [4]. These combinations can subsequently be used to help software project managers predict how well a team will perform based on the personality types of the individual team members.

The purpose of the research reported in this dissertation is to propose a series of practical approaches that aim to help software project managers with the simultaneous allocation of resources and scheduling of tasks in a way that satisfies certain objectives, including shortest possible duration, most experienced team, lowest possible cost. Specifically, the goal is to treat the problem of resource allocation and task scheduling in software projects as a special instance of the resource-constrained project scheduling problem, which is then solved using optimization techniques found

in the field of computational intelligence to generate optimal resource allocation and task scheduling solutions. Given that a software project manager may want to allocate resources and schedule tasks in a way that satisfies a number of objectives, it is necessary to make use of multiobjective optimization methods where each objective is defined in a separate function and each assumption/condition is specified in a separate constraint function. In order to reflect as accurately and realistically as possible the factors that software project managers take into account, the functions consider a variety of resource attributes and task characteristics, including

- dependencies between tasks,
- effort and skills required by tasks,
- interdependence type of tasks,
- levels of developers' experience in skills,
- productivity rates of developers,
- salary of developers,
- personality of developers, and
- communication overhead between developers.

The resulting solutions obtained from the optimization methods are expected to consist of resource allocation matrices showing which developers are assigned on each task and Gantt charts illustrating when each task is planned to be executed.

1.2 Research targets

The dissertation sets two main groups of research targets. The first group involves performing a thorough investigation of the topic of resource allocation and task scheduling in software development, and also analyzing the significance of these activities for software project managers. It is important to determine the challenges and problems that software project managers encounter, as well as the implications of these challenges and problems for software project companies. Consequently, it is necessary to conduct a comprehensive review of related research work previously carried out in the form of systematic literature reviews, surveys and proposed approaches. Additionally, is also crucial to assess the most important technical and nontechnical factors that are

or should be taken into account by software project managers in order to specify the attributes of resources and the characteristics of tasks that will be used in formulating the objectives and constraints of the optimization methods adopted in the proposed approaches.

The second group of research targets focus on the optimization methods adopted in the proposed approaches and, in particular, involve evaluating how well the various optimization methods perform in solving the problem of resource allocation and task scheduling in light of the objectives and constraints established in each approach. Taking into account the fact that the goal of the optimization methods is to generate solutions that represent the best way to correctly allocate resources and schedule tasks in a given software project, it is necessary to assess the quality of these solutions. Specifically, it is important to assess the degree to which the optimization methods are able to meet the objectives of each approach in order to examine how optimal the solutions generated are. Similarly, it is also important to evaluate the extent to which the optimization methods are able to satisfy the constraints of each approach in order to observe how valid the solutions generated are. Furthermore, it is essential to be able to compare the different optimization methods in order to decide which one is the most adequate to possibly be adopted formally in a future methodology. Therefore, it is important determine how different optimization methods compare against each other in terms of performance. Since the number of tasks/available resources may vary in size, it is also necessary to assess how different optimization methods compare against each other in terms of scalability.

1.3 Organization of the dissertation

The dissertation is organized as follows: To begin with, Chapter 2 introduces the research area of resource allocation and task scheduling in software development projects, together with a description of the main problems and current open issues that software project managers are faced with. Chapter 3 then presents a literature review of various approaches that have been proposed in the past as a way of solving the problem of resource allocation and task scheduling. Next, Chapters 4–7 describe four approaches developed during the research that adopt multiobjective optimization methods as a means to allocate resources and schedule tasks in software projects. Following this, Chapter 8 provides an overview of the area of personality psychology, focusing on the significance of using personality types in software development and, in particular, for resource allocation and team formation. This chapter also provides a literature review of previous studies

and approaches that attempt to incorporate personality types of software developers. Chapter 9 then presents the details of an approach proposing the use of multiobjective optimization methods in order to allocate resources base on the personality types of software developers and the personality profile of software professions. Finally, Chapter 10 concludes the dissertation by giving a summary of the research work carried out and recommendations for future research.

Chapter 2

Resource Allocation and Task Scheduling in Software Development

This chapter introduces the key research areas of this dissertation: allocation of resources and scheduling of tasks in software development projects. Resource allocation involves assigning a developer or a team of developers to carry out a task, whereas task scheduling involves specifying the time frame in which a developer or a team of developers will work on a task. Both together attempts to answer the question: who will work on what, and when? Generally, they are both carried out by project managers in the initial phases of a software project, specifically as part of a project manager's planning activities. However, depending on the practices followed by the software development organization and the information regarding the actual software project, the way these are carried out can vary. In some cases, a software project manager is required to allocate one or more developers to each task and schedule the tasks appropriately, whereas in other cases tasks are distributed to already predefined teams of developers. There are also cases where a project manager only needs to carry out team formation, that is, to put together a group of developers without assigning them to specific tasks.

2.1 Principles of software development

According to Reifer, software development consists of three core principles: process, product and people [5]. These principles come together in the form of a software project, where software developers (the people) use tools and methodologies (the processes) to build software systems (the products). Over the years software development has evolved 'from an art, to a craft, to a proper engineering principle' [6].

2.1.1 Process

Software development organizations establish a series of phases and procedures containing a number of activities for developers to perform. The most common steps include requirements elicitation and analysis, specification definition, architectural and analytical design, implementation and integration, testing, and maintenance. Various models exist that combine these steps into formal life cycle structures with the sole purpose of identifying and describing all the activities necessary to be undertaken throughout a software product's development. Examples of available methods include the waterfall model, the spiral model, agile software development processes (such as extreme programming, XP), and IBM's Rational Unified Process, to name a few. Due to the large number of models there have been attempts to standardize these models, as in the ISO/IEC/IEEE 12207 standard (Systems and Software Engineering – Software Life Cycle Processes) [7]. Apart from these primary process development models, there is also a variety of assistive processes aiming to support the core processes, such as those involving configuration management, documentation and quality assurance. In order to carry out the development process a range of tools and applications are required to help developers in various stages of a software product's life cycle. Some tools, known as integrated development environments (IDEs), give support for all phases in the life cycle, while others provide a phase with individual and tailored assistance, such as graphical user interface designers, compilers and debuggers.

Processes will only be useful for a software development organization if its developers follow them properly and consistently. By doing so the organization will be in a position to build a management infrastructure whereby developers will easily be able to share experiences. However, an organization must periodically review its processes using metrics and assessments in order to improve them. For this reason, the Software Engineering Institute of Carnegie Mellon University has devised the Capability Maturity Model (CMM) that allows organizations to rate the maturity of their practices [8].

2.1.2 Product

Customers and software development organizations meet together to determine the objectives and scope of the software to be built. Hence, a software product defines what is to be built and its purpose in order to help those involved come up with alternative solutions, identify possible constraints and assist in management decisions. Depending on the customer's needs, software

products come in different types, such as single applications, application suites, business and consumer products, and research and development products. In order to build a high quality software product it is recommended that organizations follow an engineering methodology because the solution selected will have to satisfy both user's needs and technical requirements [5]. The quality of a software product can be measured in terms of the degree of certain behaviours [9], for instance, performance, correctness, maintainability, robustness, reliability, availability, and portability. Reifer stresses the importance of involving the customer in the development of a product for two main reasons: First, to get as much out of them as possible in terms of knowledge (especially in situations where domain knowledge is rather technical and difficult to understand by developers), and second, to get them to participate in quality assurance activities [10]. Joint Application Development (JAD) is an example of a process that promotes such user participation.

Some of the problems concerning software product development include gold plating (where features are enhanced despite having implemented user requirements) and feature creep (where additional/unwanted functionalities are added to the product making it more complicated and less efficient). In both cases, these are likely to cause budget and schedule overruns. Another caveat of software products lies in the documentation. If not executed as a planned activity the time spent on the composition of documents may actually be wasted effort on pointless paperwork.

2.1.3 People

In software development organizations, people are considered a critical success factor because it is through their skills and experience that processes are followed to build software products. Keeping staff happy, motivated and stimulated in a solid working environment is very significant for software organizations because it can help increase levels of creativity and innovation. In addition, a developer's work is often dependent on the workforce practices of the organization, such as communication, collaboration and communication. The maturity of an organization's workforce practices can be rated using the SEI's People Capability Maturity Model (PCMM) [11].

People and their role in teams are highly important for project success, and they are even taken into account in many cost estimation models, as for example in the COCOMO model [12]. Acuña et al. characterize employees in software development organizations as 'human capital' – the most important asset of the company [2]. The more effectively this capital is managed, the more competitive benefit is achieved over other organizations. Due to the importance of people in

software development organizations, there is also a great deal of emphasis given to human resource management practices in order to effectively hire, train, motivate or appraise software developers, and also to decide on whether to retain of fire software developers.

2.1.4 Challenges and issues

In order for a software project to be successful, management of these three principles must be successful. However, due to rapid changes in the software development industry, many technical, financial, political, social and even psychological factors can prevent the constant and effective management of these principles. For example, changes in technology may force organizations to change development strategies in order to remain competitive. This in turn may require the adoption of new processes, despite the fact that employees may not be familiar with these new processes. Financial factors may also cause difficulties in managing both people and product, especially concerning project budget. Also, stress, as a psychological factor, can lead to key players resigning, while issues with trust amongst team members can be attributed to social factors [5].

A summary of these challenges are shown in Table 2.1. Regarding the process perspective of software development, one of the major obstacles facing organizations is actually finding and following a suitable process to adopt and revise accordingly. Also, the fact that software project management involves many activities any process adopted will require a specialized and most likely complex system. Concerning the product dimension of software development, the size and complexity of software systems is the foremost issue arising. Also, the volatility of requirements and possible constant changes can cause product problems. With respect to the people aspect of software development, the primary issue is actually dealing with software professionals, because it is commonly assumed that each employee needs to be handled differently as unique individuals. Furthermore, assigning people to tasks is still an open issue that remains unsolved, mainly due to the various approaches and objectives that resource allocation focuses on.

2.2 Software project characteristics

A software project has a well-defined set of objectives, the satisfaction of which leads to the successful delivery of a tangible product. Before developers can begin satisfying a project's objectives, they will need to have a collection of product requirements that have been approved by the customer. In addition, the development company and the customer must agree upon a project

Table 2.1: Summary of software development challenges and issues

Process	Product	People
• Adhering to practices [13]. • Coping with the lack of planning [14, 15]. • Working out conflicts when collaborating software companies use different development methodologies [16]. • Satisfying the need for complex project management systems [17].	• Meeting the size, speed and complexity requirements of modern systems [13, 18]. • Managing volatile user requirements (moving target phenomenon) [10]. • Remaining competitive due to the globalization and internationalization of markets [19]. • Keeping up with technology [20, 21].	• Dealing with people [10, 18, 22]. • Assigning human resources successfully [10]. • Handling the lack of coordination activities [23]. • Applying theory to practice [21, 24].

schedule and a budget for the software products to be delivered. These are usually estimated by constructing a plan of the tasks to be performed and the relationships between them [5].

In the majority of projects, work is undertaken by software development teams; according to the Juran's Pateto principle, the 'vital few' 20% of a project's resources will be used for 80% of the work, while the 'useful many' 80% of a project's resources will be used for the remaining 20% of the work. The assessment of the work, usually made by project managers, is often subjective depending on the practices adopted in assessing the work and the developers. Nevertheless, project managers are still more likely to reward developers with control and authority through performance in their work rather than given through the position that a developer holds. Also, tasks are considered unique and nonrepeating, meaning that each task in the project aims to produce a different project-specific output. Despite having specific outputs, for projects requiring innovation, allowing teams to be creative and flexible is crucial for success [25].

2.3 Software project management

Brooks stated that software and its development has inherent attributes, and the problems caused by these attributes cannot be solved by a single 'silver bullet' [26]. These attributes are:

- Complexity – high effort and cost due to uniqueness of products.
- Conformity – inconsistent, ambiguous and arbitrary requirements of clients.
- Changeability – expectation that changes are to be made without difficulty.

- Invisibility – difficulty in perceiving and physically seeing the progress of software products.

Because of these attributes, it is necessary to distinguish software project management from other types of project management in other disciplines. Software project management involves the planning, organizing, staffing, directing and controlling of software development projects [5, 6]. A software project manager is therefore required to make estimates of the amount of effort a project will require, the cost of the project, as well as the scheduling of work packages, tasks and activities. Also, project managers are expected to delegate responsibility to developers, including designating developers with authority and holding developers accountable where problems and issues arise. At the same time, project managers have a role to lead and motivate their developers in order to get the most out of their creativity and knowledge. In addition, throughout the development process, they will need to be kept up to date on the project's progress, be made aware of any problems encountered, and keep track of their development teams. All these activities are subject to a number of constraints, such as time, money, people, materials, quality and standards. Due to these constraints, software project management has witnessed various changes, as stated by Pyster and Thayer [13]. Some of these changes include

- creation and adoption of software project management standards,
- introduction of new (evolutionary) software development life-cycle models and approaches,
- credentialing of software project managers,
- usage of distributed workforces, and
- building of product lines.

To help them, software project managers make use of their own past experiences and previously acquired knowledge together with the wide range of available commercial tools and techniques (such as Project®[27], Project KickStart™ [28], Basecamp [29], MindView™ [30] and RationalPlan MultiProject [31]). A study of the impact of project management information systems by Raymond and Bergeron found that such systems improve efficiency and effectiveness with respect to project planning and control activities, as well as general project performance and overall success [19]. An example of a project management information system is described by Petter and Vaishnavi, who support the reuse of software project experiences in the form of narratives illustrated through the use of wikis [32]. However, not many of these available applications

are tailor-made for the software development industry. A survey conducted by McBride highlights this, especially for monitoring, controlling and coordination activities, in which project managers use a number of different mechanisms for a given activity but also use the same mechanism for a number of activities [20].

2.4 Resource allocation and task scheduling

During the planning activities of a software project, a project manager must allocate resources and schedule tasks by determining what work will be done, how and when it will be done, and who will do it. This consists of identifying the various products to be delivered and the activities that need to be executed, as well as estimating the effort for each activity. Hughes lists the methods available to identify a project's activities as follows [33]

- Activity-based methods, which use techniques such as brainstorming, data from past projects and work breakdown structures, to determine which activities the project will comprise.
- Product-based methods, which use practices such as product breakdown structures and product flow diagrams, to determine which products the project will contain and transform them into activities.
- Hybrid methods, which apply the work breakdown structure on the products of the project.

Due to the importance of these responsibilities, resource allocation and task scheduling normally have priority over all other activities. Furthermore, resource allocations and task schedules of a project need to be updated regularly to coincide with its current status.

The most common pieces of information that a project manager is required to provide includes a description of the tasks and their requirements, the start and finish times (and consequently, duration) of each task, the dependency relationships between the tasks, as well as the descriptions of the resources that will execute the tasks [34]. The most common representations include

- Gantt charts, which represent the activities of a project with a bar chart showing the start/finish times and duration of each task. This is the most common representation used by project managers.
- Task precedence graphs, which show a project as a network of activities where nodes denote the activities and arcs denote start/finish dates of the activities.

- Resource allocation charts, which display the work to be carried out by each resource in the form of a bar chart. Only the tasks performed by a worker are displayed for each resource.
- Programme Evaluation and Review Technique (PERT), which describes a project as a network of activities, where arcs denoted the activities and nodes denote start/finish dates of the activities.
- Critical Path Method (CPM), which computes the shortest possible duration of a project which corresponds to the longest path of an activity network.
- Earned Value Analysis (EVA), which measures project progress by comparing planned work (budgeted cost of work scheduled) with earned value (budgeted cost of work performed) using earning value rules.

However, one of the issues with these is that the representations often 'cannot model the evolutionary and concurrent nature of software development' [35]. Also, in most cases, information at the beginning of a software project is often imprecise or unavailable. As a result, a project manager's estimates may be vulnerable to mistakes, which can significantly affect the progression and overall success of a software project. Adding further complexity for project managers is the fact that allocating developers and scheduling tasks are not independent activities, which is why carrying them out as such may be considered unsuitable [35]. Allocating resources to tasks without considering how tasks are to be scheduled or scheduling tasks without taking into account how resources are to be allocated could potentially cause problems for a software project, such as availability conflicts, since both activities are affected by the same constraints. Therefore, in order to avoid the possibility of future obstacles, both activities need to be worked on simultaneously.

The most significant challenge facing project managers is the fact that resource allocation and task scheduling in software development projects is a naturally complex and computationally-intensive process. In the majority of cases, project managers will have several considerations (such as budget and time constraints) that need to be satisfied when carrying out these activities. Unlike in other manufacturing, industrial and engineering projects, a software project's workforce is solely made up of knowledge workers. As a result, software development relies heavily on the cognitive and intellectual competencies of the individuals, as well as their ability to innovate and be creative. Bearing these considerations in mind makes the allocation of resources and scheduling of tasks much harder for software project managers. What's more, project managers cannot

completely depend on using experiences from previous projects nor can they use past project information unless as weak guidelines because of the fact that software projects are different every time. The challenge is made even more difficult due to the fact that software projects are intangible in nature and labour intensive, and thus adding to the levels of complexity and uncertainty. For these reasons, the problem of resource allocation and task scheduling in software development is classed as a special case of the resource-constrained project scheduling problem (RCPSP), and therefore is considered to be an NP-hard problem. This means that large-sized instances of the problem cannot be solved to optimality by exact solution methods in reasonable time (polynomial time) [34, 36]. Project managers often struggle to use a manual approach because there are many different combinations to be examined. As a result, a brute-force, exhaustive search to find the best solution will often prove inadequate and impractical, especially if they have a limited amount of time at their disposal. Consequently, the majority of research works view the problem as an operational research problem, where proposed solutions make use of techniques that carry out combinatorial optimization of various software criteria, such as cost, duration, or number of defects. Examples include mathematical modelling methods and computational intelligence techniques, where specialized algorithms are employed to locate optimal or near-optimal feasible solutions as a means of providing better and faster support to decision makers. Several of these approaches emphasize that the allocation of developers and scheduling of tasks needs to take into account certain attributes of the available workforce, such as the capabilities and experience of developers in certain skills, as well as their cost. A comprehensive analysis of these research approaches for resource allocation and task scheduling in software development are presented in Chapter 3.

Chapter 3

Related Work

This chapter provides a review of research work carried out related to the problem of resource allocation and task scheduling for software development. It gives a description of a number of proposed approaches that attempt to solve the problem in a similar fashion as to the ones developed as part of the research presented in this dissertation. In general, this problem can be considered to belong to the field of operational research, which aims to solve decision-making problems found in many different disciplines, including natural sciences, engineering and social sciences. Operational research problems are solved by locating optimal or near-optimal solutions using a wide range of specialized methods, the most common of which include mathematical modelling approaches and computational intelligence techniques. Both these methods have been used extensively to solve the problem of resource allocation and task scheduling in software projects, but they have also been successful in solving a variety of other software engineering problems, including, regression models for estimating software project costs and effort [37], classification methods for evaluating software quality assessment [38], clustering techniques for software component categorization and retrieval from repositories [39], and optimization algorithms for the automatic and dynamic generation of test cases [40].

Approaches using mathematical modelling represent the problem of resource allocation and task scheduling in software projects as a model using variables, operators, equations, functions and other mathematical notations and concepts, and then attempt either to solve the model as an optimization problem or to use the model for prediction. Examples of mathematical modelling techniques include linear programming, statistical modelling and queuing theory. Approaches using computational intelligence techniques adopt various algorithms inspired by nature to solve

complicated and complex real-world problems like resource allocation and task scheduling in software projects. These techniques aim to achieve specific goals by imitating both individual and collective behaviours and qualities of human and other living beings with regards reasoning, logic and inference, learning and knowledge processing, in addition to reproduction and evolution. The most common computational intelligence techniques include evolutionary algorithms and swarm intelligence, artificial neural networks and fuzzy logic.

3.1 Mathematical modelling approaches

3.1.1 Linear programming

The first type of mathematical modelling concerns linear programming, which requires a linear objective function to be minimized or maximized in order to find optimal solutions to problems described by linear relationships subject to certain problem-specific restrictions [41]. Kantorovich, a Soviet mathematician during World War II, first introduced this approach as a means to solve several planning problems for the military, including how to optimally assign, schedule and transport resources based on their availability and cost, so that army expenses are reduced while enemy losses are increased. Consequently, linear programming has been considered by researchers as a suitable technique for helping software project managers in their planning activities also.

Li et al. used integer linear programming to help software development organizations cope with the pressures of limited resources and decreased time-to-market intervals by proposing two models concerning requirement scheduling and software release planning [42]. Their first model takes into account the precedence dependencies of requirements and the skills of available teams of developers to generate a project schedule for the development of requirements of a new release within the shortest possible make span, whereas their second model integrates requirements selection and software release planning of a project with a fixed deadline to maximize revenues in addition to providing an on-time delivery schedule. One of the assumptions of this attempt is that requirements are assigned to teams of developers to implement and not to individual developers. Additionally, for testing their proposed approach the authors used both example and real-world datasets. The authors do point out, however, that a mathematical model cannot stand alone as a project management decision support system since other real-world factors influence the decision-making process, such as psychological, personality and political factors.

Another methodology using linear programming is presented by Otero et al., which was developed to tackle the issue of project manager subjectivity in resource allocation [43]. The authors highlight that ineffectual resource allocation can lead to many problems for development organizations, such as 'schedule overruns, decreased customer satisfaction, decreased employee morale, reduced product quality, and negative market reputation' [44]. They therefore proposed the Best-Fitted Resource methodology that works to measure the suitability between the skills required by tasks and the skills possessed by the available resources. Project managers can then use the results from the methodology to decide on the most suitable (optimal) allocation of resources based on their capabilities. To test their approach the authors provided a small sample resource allocation scenario to 30 subjects, consisting of software engineers and project managers from the industry and also computer science students and professors from universities, and asked them to perform a ranking of the available resources based on their capabilities in the required skills. The results of this survey were then compared to the results obtained from their methodology, showing that such approach had potential in allocating resources to tasks.

Otero et al. presented another similar multicriteria decision-making methodology for software task assignment [45]. Here, they state that there is evidence that ineffective human resource project planning is the main reason that software development projects fail [46]. The methodology uses a desirability function as a means of assigning tasks to developers in cases where there are no optimally suitable developers in the existing workforce. It takes into account the capabilities of resources in skills, the required levels of expertise, as well as the level of significance of skills required by tasks and task complexities. A significant aspect of this approach is that it can be extended to take into account project-specific factors that a software project manager decides are important according to the needs of the project. An artificial case study was used to demonstrate the methodology, consisting of a scenario where a task needed to be assigned to one of ten candidate developers based on their skill assessment and associated cost with respect to the required skills of the task. On a practical level, the authors state that the approach can be adopted by software project managers using a simple spreadsheet implementation. However, no formal description of a tool is provided. Although it seems sensible to exploit the strengths of developers based on what each task requires, this is only realistically possible if the developer is available to carry out a task. The approach however does not address the issue of availability when computing the desirability

function and does not deal with task scheduling, which often influences or comes hand-in-hand with resource allocation.

3.1.2 Branch and bound methods

Branch and bound methods are a combinatorial approach that evaluate candidate solutions in order to minimize or maximize a certain objective function. They were introduced in 1960 by Land and Doig [47]. In these methods, all candidate solutions are stored at the root of a tree. Iteratively, they apply the concept of 'branching' to divide (or 'split') the search space into smaller spaces (branches) and then attempt to optimize the objective function by exploring these branches. Each branch is assessed to determine whether or not it satisfies the lower and upper bounds of the optimal solution. If a branch cannot satisfy these bounds then the solutions it contains are discarded by 'pruning' the search space they reside in.

Bellenguez and Néron proposed a multiskill project scheduling approach that considers the assignment of activities to resources possessing different levels of skill abilities [48]. Specifically in this approach, each activity requires specific skills at a fixed minimum level and has a time window for execution. Additionally, the staff available to undertake these activities possess one or more of these skills at different levels. The objective is to allocate resources so that the minimum level of skill is satisfied by the assigned developers in order to complete the project with the shortest possible make-span. Two lower bounds were used in the approach to prune the search tree. The first lower bound is based on blocks and uses a graph of compatibility to test whether a pair of activities are able to be executed simultaneously without violating the resource constraints and the precedence constraints. The second lower bound is based on energetic reasoning and detects if all the necessary parts of the activities that have to be processed in a given time interval can be executed or not. Experimental results using 180 generated data sets showed that the two lower bounds proved to be both complementary and efficient.

The authors also used a branch and bound method in another similar multiskill project scheduling approach [49], which adopts various decomposition rules at leaf nodes, as well as several branching strategies regarding maximum slack activities, stable set activities and loaded time-interval activities. The results showed that the approach works well on small and average size instances, whereas less efficiently on most of the large-sized instances. One of the drawbacks in

both these approaches is the need for each activity to be set a time window of execution, which may prove difficult for software project managers to determine at the start of the project.

3.1.3 Probabilistic modelling

Probabilistic modelling is a mathematical modelling approach that uses data (usually historical data) to forecast the conditions of different future states of a problem by calculating the probability of certain outcomes. A characteristic of this approach is that one or more of the variables in the model can be random.

Padberg presented a probabilistic project scheduling model, which focused on using scheduling strategies to help software development organizations to manage their human resources more effectively, arguing that software developers are the most valuable resources, and that software project managers need a useful scheduling support tool, as opposed to a common cost estimation tool that simply predicts the overall development effort needed to carry out a project [50]. Specifically, in the approach scheduling strategies represent, in quantitative terms, the effect of decisions regarding development costs and duration on the current state of a project. Once a strategy is fixed, it is inserted into the model, which computes a probability distribution estimating the completion time and cost by using several technical and non-technical factors, such as scheduling constraints, adopted software processes and complexity of components to be developed, as well as skills and experience of the human resources. Stochastic optimization techniques are then applied to optimize the expected duration or the cost of the project with regards to the allocated resources. It is important to model the intrinsic uncertainty that is part of the software process regarding the duration of activities and also the events that occur during a project. The author, therefore, claims that using a probabilistic approach can help deal with the fact that events in a project can occur with a particular likelihood. The approach considers a project to be broken down into components, to which only one team is assigned at any given time. An advantage to the approach is that it allows a team to interrupt their work on a component in order to rework a previously completed component. In addition, it takes into account the availability of development teams as well as the precedence relationships between components. However, overall this approach can only be applicable in software companies who have predefined teams of developers, with each team possessing the know-how to undertake the development of the component. For small-to-medium-sized companies that do not often have such luxury, this could be impractical.

Padberg later implemented his probabilistic scheduling model as a discrete simulation model for project managers to use as a tool to provide feedback and comparisons among varying strategies [51, 52] and also implemented a variation of the value iteration algorithm to generate optimal scheduling policies in the model [53, 54]. The premise of these works remains the same as in his previous approaches; that uncertainty inherent in the task durations can only allow a software project manager to create a schedule wherein the duration and cost are 'likely' to be minimized and so it is vital for software project managers to be able to apply dynamic scheduling policies.

3.1.4 Queuing theory

Queuing theory can be used as a mathematical model to simulate a system providing services to customers (human or otherwise) as they wait in line to be served. In general, this method attempts to minimize the duration and size of delays subject to constraints and, therefore, has practical application in problems such as scheduling, employee allocation, facility design and management, and traffic flow management.

Antoniol et al. used this technique in their approach concerning the allocation of resources in a large software maintenance project [55]. Specifically, the authors made use of stochastic simulations of queuing networks as an instrument to evaluate the probability that the project meets its deadline as the project is being carried out.

Jalote and Jain implemented a critical path/most immediate successor first approach to resource allocation targeting software projects that are to be developed by multiple teams across different geographically distributed time zones [56]. With a rise in the number of organizations adopting global software development, project managers face new communication and coordination issues in addition to technical and managerial problems. Therefore, they suggest a 24-hour software factory model that utilizes project task precedence graphs and available resources to satisfy three types of constraints: operational, skill and resource, in order to generate a near-optimal software project schedule with the shortest make span.

3.1.5 Constraint satisfaction

Constraint satisfaction is a method that is adopted as a means of modelling and finding solutions to combinatorial problems by imposing conditions on variables in mathematical functions that are all required to be satisfied. They feature in many artificial intelligence fields and other disciplines,

including planning, scheduling and logistics. Well-known examples of problems that can be solved using this method include map colouring, job-shop scheduling and even Sudoku puzzles. With respect to the software industry, the constraints regarding development projects predominantly concern budget and schedule and quality. Therefore, this method is adopted in order to attempt to satisfy the restrictions surrounding these issues.

Barreto et al. proposed the use of constraint satisfaction as an optimization approach to software project staffing, stating that process productivity and product quality are highly associated to the abilities of available resources [57]. The abilities taken into consideration included skills, knowledge, experience, capabilities and roles, and together with the characteristics of a project's activities and any development organization constraints, various utility functions can be maximized or minimized depending on the project manager's needs. The possible optimizers implemented consisted of most or least qualified team, cheapest team, smallest team, and best partial solution team. It is assumed that tasks are broken down into small units of work to which only one developer can be assigned. Once the software project manager decides which these tasks are, the tool performs optimization in order to locate the developer assignments that best fit the chosen utility function. The approach concentrates solely on the allocation of resources, while the starting and finishing times of tasks are known beforehand.

As an extension to their previous approach, Barreto et al. incorporated a mechanism to also handle developer productivity [58]. The authors state that the time taken to carry out a task is affected by the developer's level of productivity. Hence, the approach proposes various productivity modifiers computed based on the experience, the profession or the activity itself. A software project manager selects to apply one of these modifiers and then a new duration for each task is estimated accordingly (either increasing or decreasing it based on the developer assigned). A new utility function was subsequently implemented to enable assignments yielding the fastest team. The ability to factor in productivity is very important for software companies as the accuracy of budget and schedule estimates can be improved.

3.2 Computational intelligence techniques

3.2.1 Evolutionary algorithms

Evolutionary algorithms are a class of population-based algorithms that stem from the theory of natural evolution. They are most widely used to solve search-based problems that require

some form of optimization due to their ability to explore and exploit a problem's search space more efficiently and effectively for the purpose of locating optimal/near-optimal solutions. Consequently, evolutionary algorithms have been commonly applied directly or indirectly to the problem of resource allocation and task scheduling in software development projects. In order to find optimal/near-optimal solutions, evolutionary approaches evaluate each individual in the population, which represents a candidate solution, using an objective function that rates the fitness of solutions and checks whether they satisfy various constraints. Stronger candidates are passed into subsequent generations whereas weaker ones are discarded, leading to the detection of optimal/near-optimal solutions.

One of the earliest instances of using evolutionary algorithms for software project resource allocation and task scheduling is found in the work of Chang, Chao, Hsieh et al., who formalized a model for software project management, namely SPMNet, in the mid-90s [59]. Their approach focuses on the fact that software development organizations fail to assign the right developers to the right tasks due to the difficulties faced by project managers in handling the high level of complexity involved in finding optimal/near-optimal schedules. Their approach employs a single-objective genetic algorithm as a 'schedule optimizer' aiming to minimize the total duration and cost of a software project through a process of assigning software developers to tasks [59, 60]. One of the practical benefits of the formal software management model proposed is that it allows software project managers to track the progress of a project by working together with developers and customers. It also addresses the issue of risk management by enabling the pre-executing SPMNet and, hence, predicting the future states of a project. Over the years, this model has been significantly extended to support features to deal with additional software project management issues, such as partial assignment of developers to tasks, developer overload, and multiple project scheduling [34], in addition to developer reassignments, task suspensions and resumptions, learning, and task-specific deadlines [35].

Ge and Chang also used the schedule optimizer to implement a capability-based scheduling framework, in which task durations are calculated through system dynamics simulation that focused on the capabilities of the available personnel [61]. The authors state that it is important to consider developers' capabilities because they can influence a team's average productivity, which is determined by factors such as individual productivity, overworking, and communication overhead. Being able to simulate the effect of an assignment based on the capabilities of the developer

going to carry out a task could provide software project managers vital information at any stage during the project. However, exact details on how system dynamics simulation manages to generate task durations are not provided, which severely limits the assessment of its applicability in real-world settings.

An extension to the capability-based scheduling framework is suggested by Jiang et al., who incorporate personnel risks based on historical data during the assignment process [62]. This addition is aimed at helping software project managers identify, analyse and monitor possible risk factors arising from human activities (for example, late-in-the-day coding) and allow them to regulate resource assignment. Furthermore, the authors adapt the previous genetic algorithm to a multiobjective schedule optimizer, employing a weighted sum method to allow for trade-off solutions to be generated. Another approach using multiobjective optimization was implemented again by Ge to provide a framework for scheduling and rescheduling software projects [63]. The approach takes into account the skills and capabilities of available developers and attempts to provide an optimal project schedule based on efficiency (minimum cost and duration) and also stability factors (minimum impact of disruptions caused by rescheduling developers).

Alba and Chicano also employed genetic algorithms to develop an automated tool to allocate resources to tasks taking into account duration, resource skills, cost and global complexity [64, 65]. Their research work was centred on the fact that one of the goals of software project managers is to reduce both the cost and the duration of software development projects, even though these two goals can be conflicting. Each individual in the population is an assignment matrix representing the allocation of resources to tasks. The quality of each assignment matrix regarding cost and duration is evaluated through two objectives using the salary of each developer, the degree of dedication each developer is permitted to work on each task and the effort required for each task. The project's schedule is constructed directly as the result of which resources have been allocated to each task. As the algorithm executes, solutions converge to the optimal/near-optimal allocations and schedules. By allowing project managers to adjust weights according to the problem at hand, they have the ability to perform different scenario analyses and make better decisions regarding the software project. This is a significant feature because the importance of each criterion is subject to the software being developed within the project, thus it is reasonable to expect a software project manager in some cases to want to give emphasis on minimizing the cost of the project and

in other cases to want to focus on minimizing the project's duration, depending on which criterion he or she considers more important. One drawback to the approach, however, relates to the way that developer skills are handled. Specifically, the skills possessed by developers are treated as Boolean; either a developer possesses a certain skill or does not, and this information is used to evaluate whether the skills required by the project's tasks are satisfied in the form of a constraint. However, in reality, most project managers do not treat skills in such way but rather take into account that developers possess skills at varying levels. Therefore, the approach would make more sense to address this as part of the evaluation of objectives (that is, as an additional criterion for assigning developers to a task) rather than part of the assessment of the constraints. A comparison of several multiobjective evolutionary algorithms using various quality indicators was subsequently performed in Luna et al. [66] and Chicano et al. [67] using the same representation, that is, with each solution comprising a series of developers possessing a set of skills, which are matched against the skills required by the project's tasks. None of the experiments in this group of approaches, however, has been tested on real-world software projects. Instead they have only been applied to a collection of simulated projects, which were created by an instance generator that randomly creates a set of tasks (with associated costs and required skills) and a set of developers (with associated salaries and skills possessed). The randomness of the generated software projects may not always accurately reflect, for example, the correlation between skill set and salary of a developer where higher-skilled employees are more likely to be paid more.

In the approach proposed by Duggan et al., project managers supply the complexity of the packages to be developed (using McCabe's cyclomatic complexity measure [68]) and the proficiency (from novice to expert) of the available software engineers in each of the packages, and using a multiobjective genetic algorithm aims to find an optimal solution that minimizes the number of defects per unit of complexity and minimizes the duration of the project with a specific assignment of developers [69]. However, software project managers may find it difficult to adopt this approach because it is strictly focused on allocating resources and scheduling tasks belonging to implementation phase of a development project, and only if the project is developed using an object-oriented approach.

Kapur et al. proposed a hybrid approach, which employs integer linear programming in conjunction with genetic algorithms for resource scheduling and allocation, targeting planning product releases [70]. The authors emphasize the fact that software developers have different levels

of skills and so their goal is to help project managers assign the most qualified developers to the required tasks in order for them to achieve maximum productivity, which in turn leads to a product release offering features that maximize business value. The optimization carried out using the genetic algorithm helps software companies decide which features should be included in a particular release for its customers. The two-step method was applied to a real-world project carried out at Chartwell Technology, which specializes in developing online gaming and gambling software, demonstrating how change requests, user requirements and improvements were planned and ordered. This approach, however, can only be used for allocating and scheduling human resources for software projects developed incrementally. Ngo-The and Ruhe further develop this two-phase approach again aimed at incremental software development [71]. The authors use integer linear programming to fix an upper bound to the maximum possible achievable business value according to stakeholders' satisfaction, and then employ a genetic algorithm to evaluate this value and subsequently find an optimal/near-optimal allocation and schedule of developers to tasks in order to plan which features are to be included in each release and which are to be postponed. The approach also allocates non-human resources, such as capital, during the assignment procedure. One of the benefits of the approach, as stated by the authors, is that project managers can replan features and reschedule resources if requirements are changed or new requirements are introduced by simply using the same two-phase approach with modified inputs and parameters.

Several attempts carried out by Antoniol et al. had the sequence of execution of work packages and the assignment of teams to work packages evaluated using a hybrid of queuing simulation and a single-objective genetic algorithm [72, 73]. A shift to a multiobjective genetic algorithm was then made in the approach suggested in Gueorguiev et al., which highlights the difficulties in constructing project schedules with regards to risk [74]. The main objective of this approach focuses on the conflicting objectives of robustness and completion time, but the approach can be used implicitly for resource usage maximization. Furthermore, the adoption of queuing simulation for task staffing and optimization for scheduling tasks are also part of a later approach in Di Penta et al., where additional features are implemented to deal with fragmentation, software developer specialization, and work package dependencies [75]. Ren et al. opted for a different approach to optimizing the sequence of execution of work packages and assigning developers to tasks by

adopting a cooperative co-evolutionary method, which tries to evolve two populations of individuals simultaneously through collaboration, rather than having individuals in a single population compete against each other [76].

Yannibelli and Amandi proposed a knowledge-based genetic algorithm to aid project managers at the early stages of scheduling to staff software projects with the most effective employees [77]. Specifically, the approach uses available knowledge about employees' previous participation in projects to evaluate how effective a set of resources will be if assigned to a specific activity and how effective each individual in that set will be. With this knowledge, the algorithm attempts to find feasible and optimal project schedules satisfying the precedence relationships between the activities and the human resource requirements. An important aspect of this approach is that allocations are based not only on the skills of developers but the level of effectivity that is the result of two or more developers working together on the same task. This is an attempt to reflect real-world practices, since a software project manager may be hesitant to assign a task to a pair of developers when he or she is aware that the pair is less effective working together, even though individually the developers possess a higher level of skills than another pair of developers. It might be preferable to allocate two developers who are less skilled, but more effective working together in order to be more productive. What the authors do not make clear, however, is whether the duration of a task is specified knowing the exact number of developers to be assigned to it. What would be more flexible if this is not the case is having the duration of a task to actually shorten or stretch depending on the final level of effectivity resulting from the developers assigned.

3.2.2 Swarm intelligence

Swarm intelligence algorithms are a specific group of methods found in the field of computational intelligence that are inspired by the behaviour of biological systems found in nature, such as the flocking of birds and the schooling of fish. The aim of swarm intelligence algorithms is to mimic how each individual in the swarm acts and interacts with other individuals in its environment to achieve a common goal shared by all individuals. In particular, swarm intelligence algorithms, such as ant colony optimization and particle swarm optimization, work similarly to evolutionary algorithms by assessing the quality of the solution that each individual in the swarm represents. In the case of resource allocation and task scheduling for software development, these types of algorithms are only just now beginning to be applied, though the general goals of the approaches still

focus on minimizing the cost and duration of software projects in similar fashion to evolutionary algorithms.

Chen and Zhang recently proposed a model that combines an event-based scheduler with ant colony optimization, aiming to provide solutions consisting of reduced project costs and more stable workload assignments [78]. Essentially, the model considers that resource allocations are affected by specific events: the starting time of the project, the time when developers join or leave the project, and the time when developers are released from completed tasks. When one of these events occurs during the project, the event-based scheduler modifies the allocation of resources based on the priority given to tasks, the skill proficiency of developers, and the current availability of the developers. Then, the method goes on to construct a new schedule by using ant colony optimization, where specifically artificial ants are iteratively dispatched to build project plans. The practical benefits with this method are that it allows a software project manager to have the flexibility to pre-empt tasks, but also to be able to handle and avoid resource conflicts. Experiments were carried out on 80 artificial projects and three real-world business software projects of a departmental store and the results demonstrated that the combination of event-based scheduler with ant colony optimization was effective yielding solutions with the lowest project cost.

Xiao et al. also presented an approach using swarm optimization to allocate resources and schedule tsks in a software project [79]. The authors adopted a similar approach as Alba and Chicano [64], but instead of using a genetic algorithm to generate solutions with optimal developer assignments and project schedules they adopted ant colony optimization. They used the same objectives, that is, to minimize cost and duration, subject to the precedence relationships of tasks and skills of developers. The authors show through the results of optimization on 30 randomly generated project instances that this approach outperformed the original.

3.2.3 Fuzzy logic

Fuzzy logic is regarded as a control system for solving problems based on information that is imprecise, ambiguous, uncertain, or even missing and is used to imitate the human decision-making process on a linguistic (descriptive) rather than a numeric basis. The goal is to model the vagueness of variables that do not possess clear (crisp) distinction between its possible values. Instead, it divides the variable into (usually) overlapping (fuzzy) sets and with the use of membership functions determines the degree to which a specific value falls into each set. It has been applied

in many disciplines, such as robotics, medicine and management, where it has helped overcome subjectivity of the decision maker.

One attempt at using fuzzy logic for software project scheduling was proposed as a decision support system by Hapke et al. (1994), who claim that, due to the uncertainty of time parameters, software project managers can only approximate the durations of development activities. The fuzzy project scheduling system proposed, therefore, creates intervals representing possible durations of tasks and aims to assign software engineers to development phases taking into consideration the completion time and maximum lateness of a software project. The time criterion is cut into lower and upper bounds generating a set of optimistic and pessimistic scenarios, which are then optimized using priority heuristic rules. Because the approach only handles the minimization of the duration of projects, its applicability in the industry may be limited. The fact, however, that human resources are considered renewable resources severely increases its limitations, since it does not accurately reflect the impact that developers' capabilities can have on allocation and scheduling.

Fuzzy logic was also employed as a means for project scheduling by Callegari and Bastos (2009) in order to handle the difficulties present in pure mathematical models, for example, 'the partial loss in meaning in terms of knowledge representation'. The multicriteria resource selection method proposed employs multivalued logic and a set of inference rules to rank available resources according to their suitability to specific tasks, thus allowing project managers to assign resources to task. Specifically, a fuzzy rule matrix is constructed that stores how suitable an assignment is based on the skill level expected by a task and the current skill level possessed by the assigned developers. If-then rules then help software project managers allocate resources in order to meet the requirements of each task. One advantage of this approach is that the rules can help avoid poor utilization of developers, which is considerably important for software development companies as highly experience developers are not wasted on tasks requiring low levels of skills. However, one criticism is its inability to handle the scheduling of developers simultaneously. This is one of the few approaches that also demonstrate a prototype tool to show how a software project manager can adopt the approach in the industry.

3.3 Discussion on related work

The approaches regarding resource allocation and task scheduling for software development discussed in Section 3.1 and Section 3.2 are summarized in Table 3.1. The approaches are grouped

by the method/technique adopted and provide the goals, constraints and the type of data used in each attempt. As can be seen, the majority of attempts employ computational intelligence methods as a form of optimization, with the most popular technique being evolutionary algorithms.

Not getting the right people to do the right job at the right time can be detrimental to the success of a software project. Various techniques borrowed from several fields have been used to help avoid this through different approaches allocating resources and scheduling tasks in software projects. But despite the evolution over the years, the problem still remains unsolved largely because there is no consensus on the criteria that these research approaches need to target to create a successful resource allocation and task scheduling tool. There are several notable points regarding the approaches that need to be addressed.

Firstly, even though there have been many approaches proposed, their ability to be applied in real-world environments is not always clear. First and foremost, any approach should be accompanied with some sort of tool to show exactly how the approach could be adopted by software project managers and not provide only a description of the underlying mechanisms. Additionally, the information needed to execute any approach should be easily obtainable and measurable where necessary by software project managers, such as the dependency relationships between tasks in order to validate the feasibility of a schedule. But, for example, things like units of complexity may not be able to be provided by a software project manager, especially at initial stages of the project. Also, some attempts have put their approach to the test using simulated or artificial projects only, without obtaining results from experiments on real-world cases. This may lower a project manager's perception of the practicality of the approach.

Secondly, the majority of the research works approach the problem as a (multi)optimization problem, in that they aim to minimize/maximize several objectives, with genetic algorithms being the most prevalent of approaches. The most popular objectives involve the cost and duration of the project – two of the three dimensions of software project success – through allocating resources and scheduling tasks in such a way that the assignments yield a balance between the two.

Thirdly, some approaches consider software developers as interchangeable resources, especially when it comes to dealing with the skills required by tasks and the skills possessed by developers. Just because two developers possess the same skill, it does not mean that they will carry out a task in the same way or within the same time. Software developers are knowledge workers, and it is with this knowledge that software is built. The varying levels of skill proficiency and

experience between developers can be directly related to their salary, as well as to the time it takes to carry out a task. Therefore, in approaches trying to allocate resources and schedule tasks, it is important for software project managers to be able to factor in the variance caused by different levels of performance and productivity of developers.

Chapters 4–7 describe a series of attempts developed as part of this research work to deal with the problem of resource allocation and task scheduling in software development projects, similar to the way the approaches in this chapter attempt to solve the problem. The aim is to present the gradual evolution of a practical and applicable approach to scheduling tasks and assigning developers for projects in the software development industry. A total of four attempts are presented, which use computational intelligence techniques to optimize several software development-related criteria. These attempts focus on certain objectives, constraints and parameters that have not always been taken into account in previously proposed approaches. For example, the level of experience that developers possess in various skills has often been overlooked in previous approaches. In fact, in some approaches resources are considered interchangeable meaning that one developer is as equally competent and skilful as another. However, this does not always reflect real-world situations, since developers acquire different levels of expertise in a number of skills over time and through the application of their knowledge. Another example involves the communication between team members and the effect it has on the time taken to complete a task and the overall development time of a project. Yet, when a team of developers is required to carry out a task it is more likely that an overhead will be incurred due to the time necessary to coordinate activities and to collaborate in their work.

Skill proficiency and experience levels are not the only things that differentiate developers. Performing resource allocation and task scheduling using only these technical factors of software development means that other, nontechnical ones are neglected. Amrit argues that approaches based strictly on skills and experience may be inadequate for project managers to help them handle issues like interpersonal relationships among developers [80]. Such human, social and cultural aspects are strongly exhibited in software development companies, especially as they become more reliant on teamwork and collaboration and the emergence of distributed development. For this reason, more and more research work is being carried out that try to incorporate non-technical aspects, especially human-centric factors, involved in software development. In particular, Chapter 8 investigates the impact of the personality type of software developers and provides an overview of

the various approaches that have been proposed attempting to incorporate personality types into human resource management, either as part of team formation strategies or as part of allocation and scheduling activities. Furthermore, Chapter 9 provides a description of an approach that incorporates the nontechnical factor of personality types in an optimization method in an attempt to allocate resources based on the suitability of developers' personality type to software tasks and the level of experience in the required skills.

Table 3.1: Summary of task scheduling and developer allocation approaches

Research attempt	Goals/objectives	Constraints	Data used
Linear programming			
Li et al. [42]	– Minimize project duration – Maximize revenues	– Requirement precedence satisfaction – Team availability	– Simulated – Real-world
Otero et al. [43] Otero et al. [45]	– Maximize suitability of developers	– Skill/expertise requirements satisfaction	– Simulated
Branch and bound methods			
Bellenguez and Néron [48] Bellenguez-Morineau and Néron [49]	– Minimize project duration	– Skill/expertise requirements satisfaction – Task precedence satisfaction	– Simulated
Probabilistic modelling			
Padberg [50, 51, 52, 53, 54]	– Minimize project duration	– Skill/expertise requirements satisfaction	– Simulated
Queuing theory			
Antoniol et al. [55]	– Minimize risk of delay	– N/A	– Real-world
Jalote and Jain [56]	– Minimize project duration – Skill/expertise requirements satisfaction	– Task precedence satisfaction	– Simulated – Real-world
Constraint satisfaction			
Barreto et al. [57, 58]	– Minimize project cost – Maximize/minimize team quality – Minimize team size – Minimize project duration	– Resource requirements satisfaction – Developer availability	– Simulated

Table 3.1: Summary of task scheduling and developer allocation approaches (continued from previous page)

Research attempt	Goals/objectives	Constraints	Data used
Evolutionary algorithms			
Chang et al. [59] Chang et al. [60] Chang et al. [34] Chang et al. [35]	– Minimize project cost – Minimize project duration – Minimize amount of overtime – Resource requirements satisfaction	– Developer overtime limit satisfaction – Developer availability – Hard deadline satisfaction	– Simulated
Ge and Chang [61]	– Minimize project cost	– Developer availability – Developer overtime limit satisfaction – Task precedence satisfaction	– Simulated
Jiang et al. [62]	– Minimize project cost – Minimize project risk	– Developer availability – Developer overtime limit satisfaction – Task precedence satisfaction	– N/A
Ge [63]	– Maximize efficiency – Maximize stability	– Developer availability – Developer overtime limit satisfaction – Task precedence satisfaction	– Simulated
Alba and Chicano [64, 65] Luna et al. [66] Chicano et al. [67]	– Minimize project cost – Minimize project duration	– Developer overtime limit satisfaction – Task precedence satisfaction – Resource requirements satisfaction – Skill/expertise requirements satisfaction	– Simulated

Table 3.1: Summary of task scheduling and developer allocation approaches (continued from previous page)

Research attempt	Goals/objectives	Constraints	Data used
Evolutionary algorithms (cont.)			
Duggan et al. [69]	– Maximize project duration – Minimize software defects	– Package precedence satisfaction – Team utilization – Cross-communication overhead	– Simulated
Kapur et al. [70]	– Maximize business value	– Feature precedence satisfaction	– Simulated
Ngo-The and Ruhe [71]		– Task precedence satisfaction – Developer availability – Release deadlines satisfaction – Feature release satisfaction – Resource requirements satisfaction	– Real-world
Antoniol et al [72, 73]	– Minimize project duration	– N/A	– Real-world
Gueorguiev et al. [74]	– Minimize project duration – Minimize project overruns	– Work package precedence	– Real-world
Di Penta et al. [75]	– Minimize project duration – Minimize schedule fragmentation	– Work package precedence satisfaction – Work package assignment satisfaction – Skill/expertise requirements satisfaction	– Real-world
Ren et al. [76]	– Minimize project duration	– Work package precedence satisfaction – Resource requirements satisfaction	– Real-world

Table 3.1: Summary of task scheduling and developer allocation approaches (continued from previous page)

Research attempt	Goals/objectives	Constraints	Data used
Evolutionary algorithms (cont.)			
Yannibelli and Amandi [77]	– Maximize effectivity levels of teams	– Task precedence satisfaction – Resource requirements satisfaction	– Simulated
Swarm intelligence			
Chen and Zhang [78]	– Minimize project cost	– Task precedence satisfaction – Developer overtime limit satisfaction – Resource requirements satisfaction	– Simulated
Xiao et al. [79]	– Minimize project cost – Minimize project duration	– Developer overtime limit satisfaction – Task precedence satisfaction – Resource requirements satisfaction – Skill/expertise requirements satisfaction	– Simulated
Fuzzy logic			
Hapke et al. [81]	– Minimize project duration	– Task precedence satisfaction – Developer availability	– Real-world
Callegari and Bastos [82]	– Maximize suitability of developers	– N/A	– Simulated

Chapter 4

Approach 1: Optimizing Project Duration, Developer Experience and Developer Availability using a Weighted-Sum Genetic Algorithm

This chapter presents the first of four approaches that were developed as part of the research work described in this dissertation. The approach [83] attempts to solve the problem of resource-constrained software project scheduling by employing a weighted-sum genetic algorithm in order to generate a (near-)optimal solution where resources are allocated and tasks are scheduled so that the following criteria are satisfied: (1) the software project completes within the shortest duration possible, (2) tasks are undertaken by the most experienced developers, and (3) developers are only assigned to one task at any given time in the project to prevent availability conflicts. To satisfy these three criteria, the approach evaluates solutions with three objective functions, which consider information regarding the idle time between tasks, the experience of developers and conflicts in developers' assignments.

As previously mentioned, the problem of resource allocation and task scheduling for software development can be thought of as a particular instance of the resource-constrained project scheduling problem (RCPSP). Consequently, the approach utilizes an adapted RCPSP description that considers various attributes and requirements of project tasks and available resources specific to software development necessary to carry out multi-objective optimization.

4.1 Research questions

It is important to assess whether the use of a weighted-sum genetic algorithm method is able to generate (near-)optimal resource allocation and task scheduling solutions that satisfy the objectives

put forward in the approach. The research targets, therefore, aim to evaluate the quality of the solutions generated by the algorithm.

In particular, the first research question (RQ1.1) aims to assess how well the objective function responsible for optimizing the project duration performs in generating the (near-)optimal task schedule of a software project: **RQ1.1: How well does the project duration objective function perform in generating (near-)optimal task schedules?**

Similarly, the second research question (RQ1.2) assess the performance of the objective function responsible for optimizing the developer experience in generating the (near-)optimal resource allocation of a software project: **RQ1.2: How well does the developer experience objective function perform in generating (near-)optimal resource allocations?**

The third research question (RQ1.3) assess the performance of the objective functions when they are left to compete against each other. Because the approach adopts a weighted-sum method, assigning different weights to the objective functions means that different (near-)optimal solutions will be generated. Hence, it is important to also assess the quality of these solutions with different weight values: **RQ1.3: How well do all three objective functions perform in generating (near-)optimal resource allocation and task schedules when competing against each other at different weight values?**

To help answer these research questions, a number of experiments are conducted using several software project instances.

4.2 Problem description

A software project consists of m tasks, denoted by the set $T = \{t_1, t_2, \ldots, t_m\}$. Each task, t_i, is associated with a duration, t_i^{duration}, which corresponds to the length of time that it requires to be completed. The execution of the tasks in the project is subject to certain logical relationships that may exist between the tasks. These relationships are specified in the set of dependencies, D, which consists of pairs of tasks such that $(t_i, t_j) \in D$ if the execution of task t_j depends on the execution of task t_i. In the approach, it is assumed that all task dependencies follow a finish-to-start relationship, that is, a task can only start executing once all its predecessor tasks have finished executing. A task precedence graph (TPG) consisting of nodes and edges can be used to help depict these relationships, where the nodes and edges represent the tasks and dependencies

between tasks, respectively. Figure 4.1 illustrates an example task precedence graph of a software project consisting of eight tasks and ten dependencies.

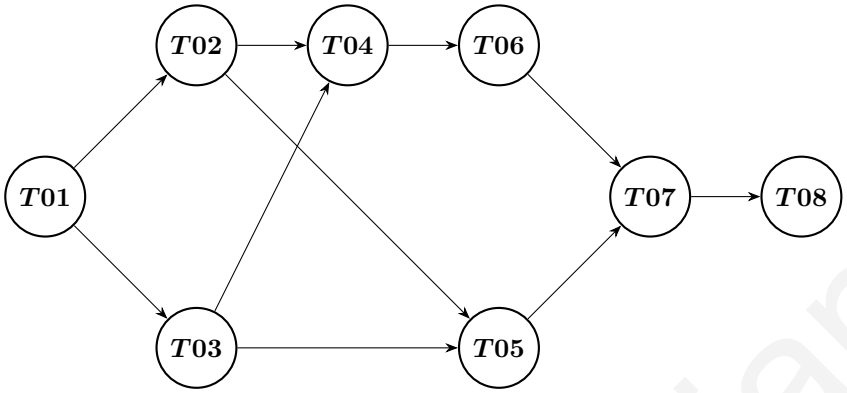


Figure 4.1: An example TPG for a software project

A software development company has n resources (developers) available to carry out a project, expressed by the set $R = \{r_1, r_2, \ldots, r_n\}$. A project requires a set, $S = \{s_1, s_2, \ldots, s_p\}$, of p skills to be used by these available developers in order to carry out its tasks. The approach considers that a task may require one or more of these skills in order to be carried out; this information is represented by the $m \times p$ logical matrix $TREQ = [treq_{ik}]$, where $treq_{ik} = 1$ if task t_i requires skill s_k or $treq_{ik} = 0$ if task t_i does not require skill s_k. Also, the approach assumes that a developer may possess experience in one or more of these skills. The level of experience of each developer in each skill is represented by the $n \times p$ matrix $LEXP = [lexp_{jk}]$, where $lexp_{jk}$ denotes the level of experience that developer r_j possesses in skill s_k. The level of experience can take a value in the range $[0, 1]$, where a value of 0 means that a developer has no experience in a skill, whereas a value of 1 means that a developer is highly experienced.

The approach uses these definitions to employ a weighted-sum genetic algorithm in order to generate (near-)optimal resource allocation and task schedule solutions that satisfy the aforementioned objectives.

4.3 Weighted-sum genetic algorithm method

Genetic algorithms are a computational intelligence technique introduced by John Holland in 1975 [84] that attempts to solve optimization problems by simulating the process of natural selection using concepts from biological evolution. To begin with, an initial population (or generation) of individuals is randomly generated. These individuals represent candidate solutions to

the problem being solved. The goal is to iteratively improve these individuals with genetic operators, eventually leading to the location of the individual that represents the best or (near-)optimal solution to the problem. To do this, the fitness of every individual is evaluated using some measure/criterion related to the problem being solved, which is expressed in the form of an objective function. Essentially, the fitness value of an individual calculated by the objective function determines how good its corresponding candidate solution is at solving the problem. Once the fitness of all individuals have been evaluated, the genetic algorithm proceeds to use a selection operator to chose the parents from the current population to populate the next generation with offspring individuals. Because genetic algorithms resemble the concept of 'survival of the fittest', selection takes into account the fitness of individuals; those that are fitter are more likely be selected as parents, meaning their good characteristics are passed on to the offspring in the next generation, while those that are less fit are discarded. Offspring individuals are created by recombining parent individuals with a crossover operator. In addition, offspring individuals are altered using a mutation operator in prospect of increasing the fitness of the offspring further. After the new generation is formed, the individuals of the population are again evaluated, and the selection, crossover and mutation operators are applied once more to form a new generation. This process is repeated for a fixed number of iterations or until no improvement to the individuals of the population is observed. Ultimately, with help from the genetic operators, the algorithm guides individuals to converge to the (near-)optimal solution. The steps are presented in detail in Algorithm A.1.

In some cases, certain problems require multiple criteria to be satisfied. Therefore, the problem becomes a multi-objective optimization problem where individuals need to be tested against multiple objective functions. One way to solve these problems is to adopt a weighted-sum genetic algorithm method, where the objective functions that are used to assess the fitness of an individual are aggregated into one combined objective function using predefined weights. The weights symbolize the significance of each objective function so that the fitness of an individual in the population is calculated by summing the product of each objective function with its corresponding weight. If an individual has a greater fitness in an objective that is considered more important, then it will stand a higher chance of being selected. The use of a weighted-sum genetic algorithm method is considered appropriate in this approach, since each candidate solution (resource allocation and task schedule) needs to be able to satisfy a number of objectives.

4.3.1 Representation and encoding

In this approach, each individual in the population corresponds to a resource allocation and task schedule solution. In order to assess the fitness of an individual, the objectives require two pieces of information: (1) task schedule information indicating when each task is scheduled to be executed, and (2) resource allocation information denoting which developers are assigned to each task. Thus, to represent these pieces of information, the genetic algorithm uses a single mixed-type array of length m, where each element, i, in the array contains the two necessary pieces of information for each task, t_i. Specifically, the task schedule information is represented by a positive, nonzero integer that symbolizes the scheduled start time, t_i^{sched}, of task t_i and the resource allocation information is represented by a bitset $B^i = \{b_1^i, b_2^i, \ldots, b_n^i\}$ that symbolizes which of the n developers are assigned to each task. Thus, if bit b_j^i in the bitset has a value of 1, then this means that developer $r_j \in R$ is assigned to work on the corresponding task. If the value of the bit is 0, then the developer is not assigned to work on the task. The t_i^{assigned} developers that are selected to work on task t_i are denoted by the set $A^i = \{r_j \mid \forall r_j \in R \wedge b_j^i = 1\}$. Figure 4.2 gives an example of an individual representing the task scheduling and developer allocation for a software project containing four tasks ($T01$–$T04$) and five available developers ($R01$–$R05$). In the example, task $T01$ is scheduled to start at $t_1^{\text{sched}} = 1$, with developers $A^1 = \{R01, R03\}$ assigned (since $b_1^1 = b_3^1 = 1$). Task $T02$ is scheduled to start at $t_2^{\text{sched}} = 11$, with developer $A^2 = \{R04\}$ assigned (since $b_4^2 = 1$). The complete allocation of resources for this project can be given by the set $A = \{\{R01, R03\}, \{R04\}, \{R02, R05\}, \{R03, R04\}\}$.

$\boldsymbol{T01}$		$\boldsymbol{T02}$		$\boldsymbol{T03}$		$\boldsymbol{T04}$	
1	10100	11	00010	16	01001	31	00110

Figure 4.2: Example of the representation and encoding of individuals used in Approach 1

4.3.2 Population initialization

Individuals in the population are initialized randomly. However, both pieces of information are subject to certain feasibility constraints. For each task, the task schedule information of an individual is randomly generated with an integer value greater than zero. The bitsets in the developer allocation information of an individual is also randomly generated in a way that at least one developer is assigned to each task (that is, at least one bit in a task's bitset is set to a value of 1).

4.3.3 Optimization function

The optimization function that the approach adopts in order to help guide the algorithm to generate (near-)optimal resource allocation and task scheduling solutions is given in Equation (4.1).

$$\underset{\forall x \in POP}{\text{Maximize}} \quad \mathcal{F}(x) = (w_1 \times \mathcal{F}_{\text{duration}}(x)) + (w_2 \times \mathcal{F}_{\text{experience}}(x)) + \mathcal{F}_{\text{assignment}}(x), \tag{4.1}$$

where $0 \leq w_i, w_2 \leq 1$ and $w_i + w_2 = 1$. The fitness of each individual x in the population POP is assessed using three objective functions relating to the duration of the project, $\mathcal{F}_{\text{duration}}(x)$, the experience of the developers assigned, $\mathcal{F}_{\text{experience}}(x)$, and the conflicts in the availability of developers, $\mathcal{F}_{\text{assignment}}(x)$.

By adopting a weighted-sum approach for optimizing multiple objectives, a project manager can associate different levels of importance to either the project duration or developer experience criteria so as to allow the guidance of the genetic algorithm based on his or her preference. For instance, a project manager may want to focus primarily on the construction of a project schedule with the shortest possible duration and secondarily on the experience of employees. In such a case, a higher weight value will be assigned to the project duration objective function and a lower weight value will be assigned to the developer experience objective function. If the most experienced developer is assigned to work on two parallel tasks, which leads to a conflict, then preference will be given to keeping the duration of the project the same but assigning the next most experienced developer to either one of the tasks. Conversely, if a project manager prefers to have a team that is the most experienced and gives lower priority to project duration, then in case of a conflict, the project schedule will grow in duration so that the most experienced developer(s) still remain assigned to the tasks causing the conflict. The developer assignment objective function is essentially always assigned a weight value of one in order to keep its importance in finding feasible solutions regardless of the importance of the other two objective functions.

4.3.3.1 Objective functions

4.3.3.1.1 Project duration objective function

The first objective in this approach relates to generating resource allocation and task schedule solutions in a way where a software project completes within the shortest duration possible. This objective is specified in the project duration objective function, which aims to minimize the duration of a software project by assessing the fitness of individuals in the population based on the

degree to which unnecessary delays exist between tasks. Hence, the function only requires the task scheduling information regarding the start time of each task.

To begin with, for each individual x in the population, the earliest start time t_i^{start} of each task t_i is calculated using

$$t_i^{\text{start}} = \begin{cases} 0, & \text{if } \nexists t_j \text{ such that } (t_j, t_i) \in D \\ \max\{t_j^{\text{finish}} \mid (t_j, t_i) \in D\} + 1, & \text{otherwise} \end{cases}, \tag{4.2}$$

where the scheduled finish time of task t_j is computed based on its scheduled start time and its duration as

$$t_j^{\text{finish}} = t_j^{\text{sched}} + t_j^{\text{duration}}. \tag{4.3}$$

If task t_i does not have any predecessors, then t_i^{start} is given a value of zero because it can start as soon as the project starts. Otherwise, if task t_i depends on task t_j (that is, $(t_j, t_i \in D)$), then t_i^{start} is calculated based on the value corresponding to the finish time t_j^{finish} of task t_j. In the case where task t_i has more than one predecessor, then the value given to t_i^{start} is calculated based on the finish time of the predecessor task that finishes the latest. Next, the number of idle time units elapsing between the task's scheduled start time (based on the task scheduling information in the individual) and its earliest start time (based on the finish time of its predecessors) is calculated as

$$t_i^{\text{idle}} = t_i^{\text{sched}} - t_i^{\text{start}}. \tag{4.4}$$

Figure 4.3 illustrates an example of how the idle time is calculated. Assuming that task $T03$ depends on tasks $T01$ and $T02$, then the idle time will be calculated by using the finish time of $T01$ since it is the predecessor of $T03$ that finishes the latest. Consequently, the idle time for $T03$ is $16 - 10 = 6$ days.

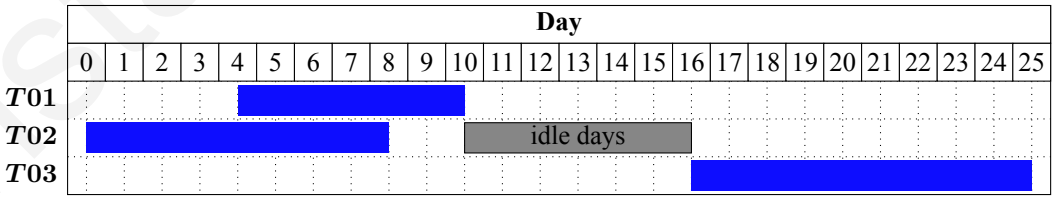


Figure 4.3: Example of the computation of task idle days

The conditional equation in Equation (4.5) is then used to calculate the overall delay in execution for each task, taking into account dependency violations caused in cases where a task is

scheduled to start before one of its predecessor tasks. Specifically,

$$t_i^{\text{delay}} = \begin{cases} 0, & t_i^{\text{sched}} < t_i^{\text{start}} \\ \dfrac{1}{1 + t_i^{\text{idle}}}, & t_i^{\text{sched}} \geq t_i^{\text{start}} \end{cases}. \tag{4.5}$$

If the scheduled start time of task t_i causes a dependency violation, then t_i^{delay} is given a value of zero. Otherwise, t_i^{delay} is calculated using the value of t_i^{idle}. The smaller the idle time, the higher the value of the t_i^{delay}. Ideally, a task should be scheduled to start immediately after its latest predecessor finishes with no delays so that $t_i^{\text{idle}} = 0$ and t_i^{delay} taking a maximum value of one. The total sum of delays obtained for all tasks using Equation (4.5) are then averaged over the number of tasks, as shown in Equation (4.6), in order to give the final value of the project duration objective function of an individual in the population.

$$\mathcal{F}_{\text{duration}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{delay}}. \tag{4.6}$$

The higher the partial values of t_i^{delay}, the fitter the individual with regards this objective due to more tasks starting as early as possible in the project without delay. It should be noted that the objective function does not compute the actual duration of the project, but as previously mentioned, it computes the degree to which there are unnecessary delays between tasks. The objective function essentially assesses the lateness of a task in the sense that a task that is scheduled to start after its earliest start time will also finish later than it is supposed to. Hence, the objective function attempts to minimize this 'lateness' and, consequently, the duration of the project.

4.3.3.1.2 Developer experience objective function

The second objective concerns generating resource allocation and task schedule solutions in a way where tasks are undertaken by the most experienced developers. This objective is defined in the developer experience objective function, which aims to maximize the experience of assigned developers by calculating the fitness of individuals in the population based on the level of experience that developers possess in the various skills required by the tasks of the project. Hence, the function only makes use of the resource allocation information regarding which developers are assigned to each task.

In this approach, the level of experience in a skill is not considered cumulative among developers, and therefore is not presented as simply the summation of the experience levels of all

developers assigned to a task. In order to calculate the total experience, $treq_{ik}^{\text{experience}}$, of the developers assigned to carry out task t_i requiring skill s_k, Equation (4.7) takes the experience level of the most experienced developer assigned and adds to that the mean level of experience of all assigned developers regarding skill s_k.

$$treq_{ik}^{\text{experience}} = \max\{lexp_{jk} \mid \forall r_j \in A^i\} + \frac{1}{t_i^{\text{assigned}}}\text{sum}\{lexp_{jk} \mid \forall r_j \in A^i\}, \tag{4.7}$$

where A^i consists of the t_i^{assigned} developers assigned to task t_i. In this way, the objective function helps assign highly experienced developers to a task and simultaneously prevents the assignment of developers without experience in the skills required (that is, non-contributors) through lowering the average experience of the team. The total experience possessed by the assigned developers of a task is then calculated by averaging the experience of all the task's required skills as given by:

$$t_i^{\text{experience}} = \frac{1}{t_i^{\text{skills}}} \sum_{k=1}^{p} treq_{ik}^{\text{experience}}[treq_{ik} = 1], \tag{4.8}$$

where t_i^{skills} indicates the number of skills required by task t_i. Last, the total experience of assigned resources obtained for all tasks is then averaged over the number of tasks, as shown in Equation (4.9), in order to calculate the value of the developer experience objective function for an individual in the population.

$$\mathcal{F}_{\text{experience}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{experience}}. \tag{4.9}$$

The higher the partial values of $t_i^{\text{experience}}$, the fitter the individual with regards this objective due to the overall experience levels of the team of developers assigned to work on each task being higher.

4.3.3.1.3 Assignment validity objective function

The approach's third objective involves generating resource allocation and task schedule solutions so that developers are only assigned to one task at any given time in the project so as to prevent availability conflicts. Its purpose, therefore, is to ensure that the algorithm locates feasible solutions. The objective is defined the assignment validity objective function, which aims to minimize the number of instances where the same developer has been assigned to two or more tasks that have been scheduled to execute (in part or wholly) simultaneously by evaluating the fitness of individuals in the population based on the degree to which conflicts exist in developers' assignments. To carry out the evaluation, the function requires both the task scheduling information and the developer allocation information.

To start with, the algorithm calculates the fraction representing the number of time units that an available developer r_j been assigned with conflicts, $r_j^{\text{conflicts}}$, over the total number of assigned time units, $r_j^{\text{totassigned}}$. Then, the algorithm uses Equation (4.10) to calculate the ratio a developer's valid assignments.

$$r_j^{\text{assignment}} = 1 - \frac{r_j^{\text{conflicts}}}{r_j^{\text{totassigned}}}. \tag{4.10}$$

After the valid assignment ratio has been computed for each developer separately, the average ratio of all developers for an individual in the population is then calculated as the final value given to the objective function given as

$$\mathcal{F}_{\text{assignment}}(x) = \frac{1}{n}\sum_{j=1}^{n} r_j^{\text{assignment}}. \tag{4.11}$$

The higher the partial values of $r_j^{\text{assignment}}$, the fitter the individual with regards this objective due to developers having fewer number of conflicts in their assignments.

4.3.4 Genetic operators

The attempt uses a roulette wheel procedure to select two individuals as parents for recombination. Individuals with higher fitness (that is, higher value of $\mathcal{F}(x)$ will have a higher probability of being selected as a parent. The two winning individuals of the roulette wheel procedure are then combined using a single-point crossover operator. To determine the position at which crossover will occur between the two parents, a series of random integers is generated, first, to select the task at which the crossover will occur, and second, to then determine on which one of the two pieces of information the crossover will take place. In this way, crossover will be applied either right after the task scheduling information or randomly between two bits in the bitset of the developer allocation information. The two offspring produced as a result of crossover then undergo a mutation operation. Similarly, a random integer is again generated to determine the task whose information will be mutated. Then, another random integer is generated to decide which piece of information will be mutated. If the task scheduling information is selected, then a polynomial mutation operator is applied on the integer representing the start day. On the other hand, if the developer allocation information is selected, then a bit flip mutation operator is applied to one of the bits in the bitset randomly.

4.4 Experiments

A series of three experiments were carried out to help answer the research questions posed in Section 4.1 concerning the performance of the objective functions. The first experiment was carried out to help answer research question RQ1.1 by setting only the project duration objective function active, $\mathcal{F}(x) = \mathcal{F}_{\text{duration}}(x)$, in order to examine whether the objective function guides the algorithm to generate (near-)optimal solutions with the shortest project duration based on the scheduling of tasks. The second experiment was conducted to help answer research question RQ1.2 by having only the developer experience objective function active, $\mathcal{F}(x) = \mathcal{F}_{\text{experience}}(x)$, in order to observe whether the objective function is able to guide the algorithm to generate (near-)optimal solutions with the highest experience based on the allocation of resources. The third and final experiment was used to help answer research question RQ1.3 by applying all three objective functions using Equation (4.1), in order to assess whether the objective functions are able to guide the algorithm to generate (near-)optimal solutions with both the shortest duration and highest experience with different weight values.

The experiments used two projects instances that were constructed using the TPG depicted in Figure 4.4, which was also used by Chang et al. [34]. The first project instance ($P1A$) consisted of ten tasks ($T01$–$T10$) of the TPG, whereas the second project instance ($P1B$) contained all 15 tasks ($T01$–$T15$) of the TPG. All tasks dependencies follow a finish-to-start relationship with their predecessor tasks.

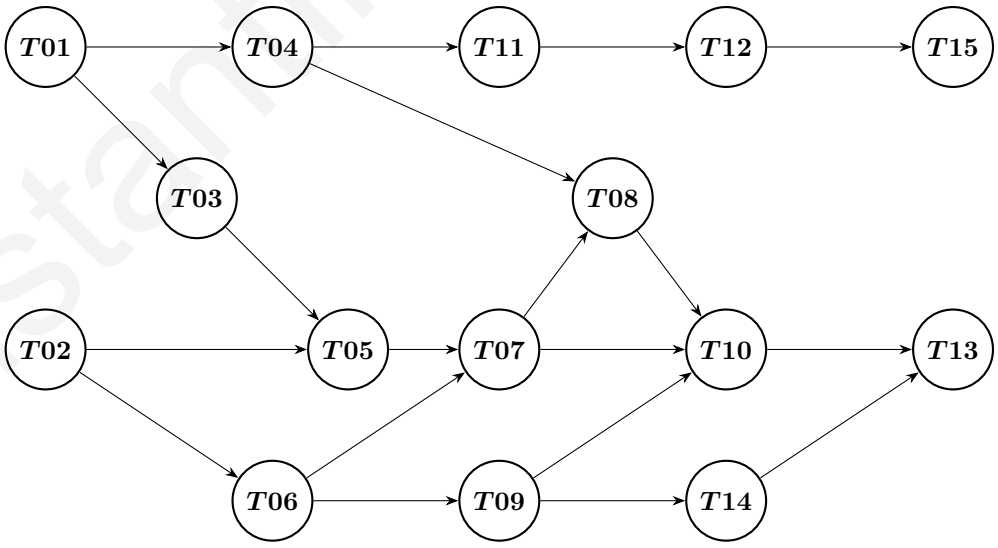


Figure 4.4: The TPG from which the two project instances used for experiments in Approach 1 were created

Table 4.1: Duration and skills required for the tasks of the two project instances used for experiments in Approach 1

		Required skill				
Task	**Duration**	$S01$	$S02$	$S03$	$S04$	$S05$
$T01$	10	✓		✓		
$T02$	15			✓	✓	
$T03$	20					✓
$T04$	10	✓		✓		
$T05$	15		✓	✓	✓	
$T06$	20		✓	✓		
$T07$	10		✓		✓	
$T08$	15	✓				✓
$T09$	20			✓	✓	
$T10$	20			✓		✓
$T11$	10	✓	✓			
$T12$	15			✓		✓
$T13$	20				✓	✓
$T14$	25		✓			✓
$T15$	15				✓	✓

Table 4.2: Experience of available developers in the skills required by the two project instances used for experiments in Approach 1

	Required skill				
Developer	$S01$	$S02$	$S03$	$S04$	$S05$
$R01$	0.0	0.0	0.4	0.8	0.0
$R02$	0.2	0.0	0.4	0.0	0.0
$R03$	0.0	0.8	0.0	0.0	0.0
$R04$	0.0	0.0	0.4	0.8	0.6
$R05$	0.0	0.6	0.0	0.0	0.0
$R06$	0.0	0.6	0.4	0.8	0.0
$R07$	0.0	0.4	0.4	0.0	0.0
$R08$	0.0	0.6	0.6	0.8	0.0
$R09$	0.0	0.2	0.4	0.0	0.0
$R10$	0.6	0.4	0.0	0.0	0.6

The duration and the set of skills required for each task of Figure 4.4 are given in Table 4.1. The project instances require a total of five skills ($S01 - S05$), with some tasks requiring two or more skills in order for them be executed. In addition, Table 4.2 provides the degree of experience that the ten available developers ($R01 - R10$) possess in each of the five skills required by the project. As can be seen, the majority of developers possess experience in more than one skill.

The genetic algorithm was run 30 times for each project instance in each experiment for 5000 iterations. The population of each run consisted of 100 individuals. Execution time ranged between 17 s and 5 min, depending also on the size of the project. It should be noted that genetic algorithms

are random in nature with respect to initialization and application of genetic operations. Because this randomness affects convergence, that is, the number of iterations required to find the (near-)optimal solution, the time taken to complete an execution will naturally vary. Hence, any figures regarding execution time can only give a very rough indication of the overall behaviour of the genetic algorithm in terms of performance.

4.5 Results and discussion

4.5.1 RQ1.1: How well does the project duration objective function perform in generating (near-)optimal task schedules?

The first experiment involved executing the genetic algorithm on the two project instances to evaluate only the project duration objective function. This was done to assess that generated solutions contained no unnecessary delays between tasks and no dependency violations. Out of the 30 runs, 23 (77%) runs were able to find the (near-)optimal solution for project $P1A$, whereas 14 runs (47%) for project $P1B$. An example of the convergence of the genetic algorithm for both test projects can be seen in Figure 4.5.

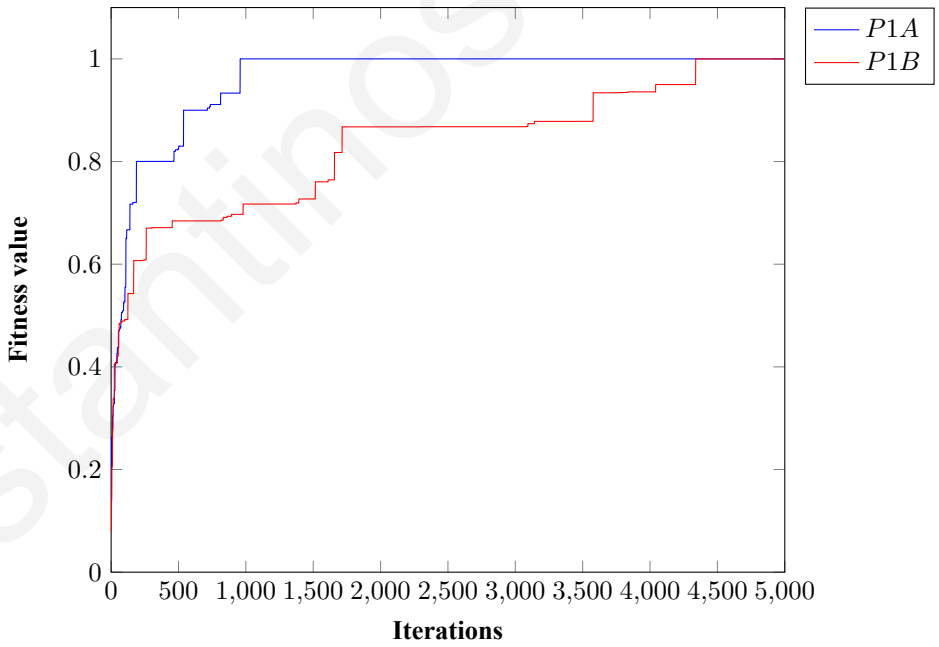


Figure 4.5: Graph showing the iterations required for the genetic algorithm to converge to the (near-)optimal solutions found for projects $P1A$ and $P1B$ with only the project duration objective function active in Approach 1

For project $P1A$ the (near-)optimal solution was found around 1000 iterations, whereas for project $P1B$, as expected, required a higher number of iterations (roughly 4300). Furthermore, construction of the corresponding (near-)optimal project Gantt charts correctly shows that the shortest possible duration for project $P1A$ is 90 days and for project $P1B$ the shortest make-span is 110 days (Figure 4.6).

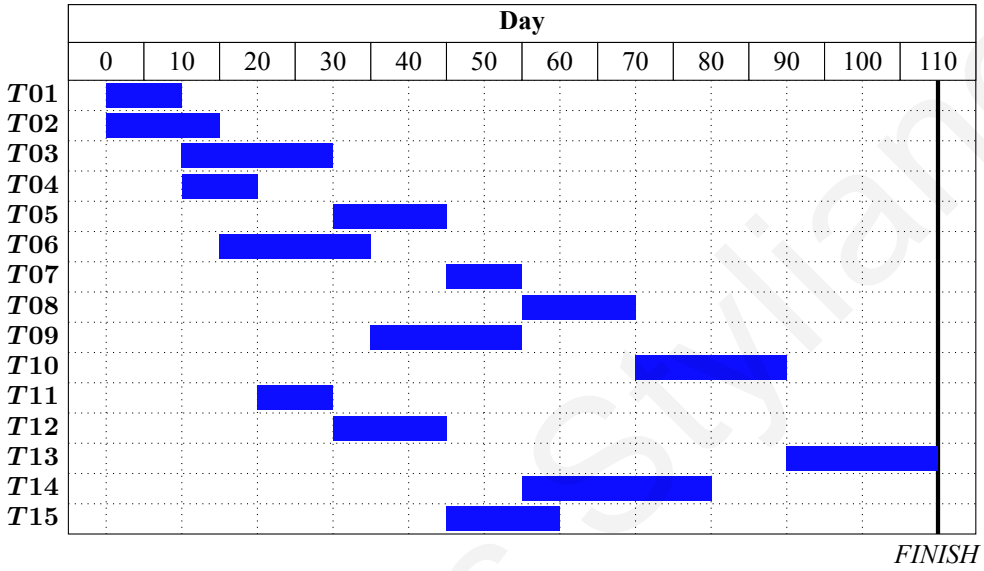


Figure 4.6: Gantt chart of the optimal solution generated for project $P1B$ with only the project duration objective function active in Approach 1

4.5.2 RQ1.2: How well does the developer experience objective function perform in generating (near-)optimal resource allocations?

The second experiment examined whether the algorithm was able to find the (near-)optimal resource allocation by only evaluating the developer experience objective function for individuals. All 30 runs were successful in assigning the most experienced developers or team of developers to each task in both project instances in such a way that all skills were satisfied and no idle or surplus developers were used. Table 4.3 shows the (near-)optimal resource allocation matrix generated for project $P1A$ displaying which of the available developers will staff each task. Out of the ten available developers only four ($R04$, $R06$, $R08$ and $R10$) are selected to carry out the project.

Table 4.3: Resource allocation matrix of the optimal solution generated for project $P1A$ with only the developer experience objective function active in Approach 1

	Task									
Developer	$T01$	$T02$	$T03$	$T04$	$T05$	$T06$	$T07$	$T08$	$T09$	$T10$
$R04$			✓							✓
$R06$							✓			
$R08$	✓	✓		✓	✓	✓			✓	
$R10$	✓			✓				✓		

4.5.3 RQ1.3: How well do all three objective functions perform in generating (near-)optimal resource allocation and task schedules when competing against each other at different weight values?

The final experiment was carried out to investigate the behaviour of the genetic algorithm when all three objectives are active in the evaluation of individuals. Since the objective functions are considered to be competing against each other, each objective function is multiplied by a preference (weight) as explained previously. To begin with, greater preference was given to project duration ($w_1 = 0.9$) than to developer experience ($w_2 = 0.1$). The results obtained showed that with all three objective functions active, the genetic algorithm was able to find the (near-)optimal schedule successfully in 17 runs (57%) for project $P1A$ and in eight runs (27%) for project $P1B$. Given these specific weight preferences, in order to avoid availability conflicts, (near-)optimal solutions generated by the genetic algorithm assign the next best developers in terms of experience to parallel tasks while keeping the project duration to a minimum. Table 4.4 gives an example of the resource allocation matrix of the (near-)optimal solution found for project $P1A$.

Table 4.4: Resource allocation matrix of the optimal solution generated for project $P1A$ with all objective functions active ($w_1 = 0.9$ and $w_2 = 0.1$) in Approach 1

	Task									
Developer	$T01$	$T02$	$T03$	$T04$	$T05$	$T06$	$T07$	$T08$	$T09$	$T10$
$R02$	✓			✓						
$R04$										✓
$R06$					✓		✓			
$R08$		✓	✓			✓			✓	
$R10$	✓			✓				✓		

Conversely, a higher preference was given to developer experience ($w_2 = 0.9$) and a lower preference to project duration ($w_1 = 0.1$). This was done to examine whether the genetic algorithm

was able to increase the project duration to accommodate keeping the most experienced developers assigned. The results obtained showed that the algorithm found it difficult to reach a (near-)optimal solution. Specifically, executions carried out for project $P1A$ managed to locate (near-)optimal solutions three times (10%), whereas once (3%) for project $P1B$. As expected, the algorithm, draws its attention to the developer experience objective function causing the project duration to be prolonged. An example of the evolution of the three objective functions for one of the runs for project $P1A$ in this experiment is shown in Figure 4.7.

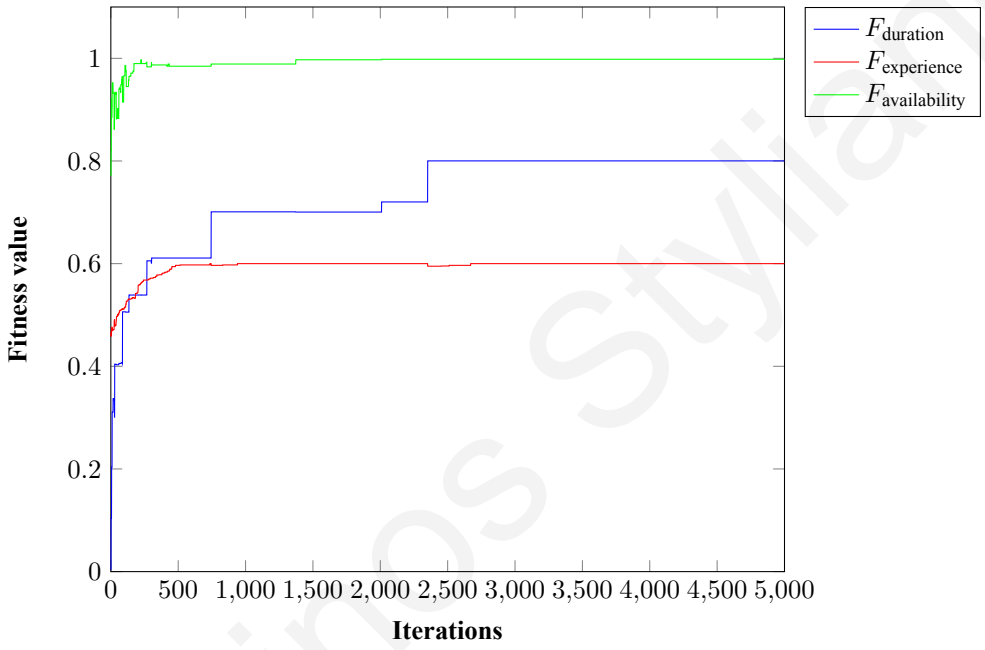


Figure 4.7: Evolution of objective functions of a run conducted for project $P1A$ with all objective functions active ($w_1 = 0.1$ and $w_2 = 0.9$) in Approach 1

As can be seen in the figure, the developer experience objective function converges to the maximum possible value (0.6 in this case), while the project duration objective function remains at much lower values since there is a lower weight preference assigned.

The corresponding Gantt chart of the solution is presented in (Figure 4.8). Notably, there is a delay in the start of task $T02$ by 15 days (it should start as soon as $T04$ completes). Furthermore, a minor dependency violation of the order of one day can be observed for task $T08$.

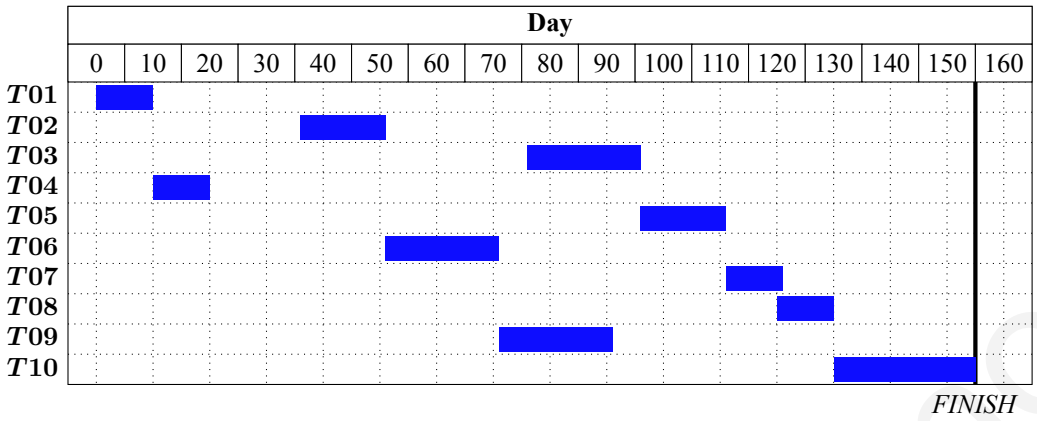


Figure 4.8: Gantt chart of a solution generated for project $P1A$ with all objective functions active ($w_1 = 0.1$ and $w_2 = 0.9$) in Approach 1

4.6 Summary

The results obtained when using only one of the objective functions showed that the weighted-sum genetic algorithm method is capable of finding (near-)optimal solutions in the majority of cases. However, when all objective functions are active the algorithm has difficulties in reaching (near-)optimal solutions, especially when assigning a greater preference to developer experience over project duration. Through observation of a number of executions, it was noticed that in this approach the algorithm is not able to reduce idle gaps between tasks nor is it able to produce feasible conflict-free schedules and allocations given the current set up. For this reason, several modifications to the approach were required, particularly with regards the optimization function, in order to improve the generation of (near-)optimal solutions and to avoid the generation of infeasible solutions. These modifications are presented in Chapter 5, which shows the introduction of constraints in the optimization function.

Chapter 5

Approach 2: Optimizing Project Duration and Developer Experience using a Weighted-Sum Particle Swarm Optimization Algorithm

Based on the deficiencies of the approach described previously in Chapter 4, a new approach [85] was constructed aiming to solve the resource-constrained software project scheduling problem by providing better and more valid solutions. Specifically, the objectives of this approach mirror the fist two objectives of the previous approach, that is, to allocate resources and schedule tasks so that: (1) the software project completes within the shortest duration possible, and (2) tasks are undertaken by the most experienced developers. However, the availability of developers is now correctly treated as a constraint of the problem (rather than an objective to be optimized), the purpose of which is help the algorithm avoid the generation of infeasible solutions where developers are mistakenly assigned to carry out more than one task simultaneously. In addition, several other constraint functions are introduced to help in the generation of higher quality, feasible solutions.

The mechanism used for optimization also differs in this approach. Specifically, this attempt employs a weighted-sum particle swarm optimization algorithm instead of a weighted-sum genetic algorithm to generate solutions for the approach.

5.1 Research questions

The primary research target is to investigate whether the weighted-sum particle swarm algorithm manages to produce acceptable and (near-)optimal resource allocation and task schedule solutions for software projects based on the new assumptions and criteria set in the approach.

Specifically, the first research question (RQ2.1) aims to evaluate the quality of the solutions generated by the algorithm in terms of feasibility: **RQ2.1: How well do the constraint functions**

perform in helping the algorithm generate feasible resource allocation and task scheduling solutions?

The goal of the second research question (RQ2.2) is to determine the quality of the solutions generated in terms of optimality: **RQ2.2: How well do the objective functions perform in helping the algorithm generate (near-)optimal resource allocation and task scheduling solutions?**

A series of experiments using a number of software project instances ranging in size and complexity were set up to help with these investigations.

5.2 Problem description

The approach uses the same definition of the resource-constrained software project scheduling problem presented previously in Chapter 4, and adopts the corresponding notations and conventions provided in Section 4.2.

5.3 Weighted-sum particle swarm optimization algorithm method

The Particle Swarm Optimization (PSO) algorithm is a type of swarm intelligence algorithm from the field of evolutionary computation. A swarm consists of a number of particles, each representing a possible solution to the problem under examination. Iteratively, the algorithm then attempts to discover the (near-)optimal solution to the problem through the cooperation and collaboration between these particles by exploring and exploiting the problem's search space. The procedure for particle swarm optimization is outlined in Algorithm A.2.

5.3.1 Representation and encoding

The representation used to solve the problem is unchanged from the previous approach described in Section 4.3.1. Each solution contains information regarding the start time of each task as an integer number and the assignment of developers as a bitset.

5.3.2 Swarm initialization

The particles of the swarm are initialized in the same fashion as in the previous approach presented in Section 4.3.2, keeping in mind that at least one developer must be assigned to each task.

5.3.3 Optimization function

The optimization function used in this second approach is modified as

$$\underset{\forall x \in SWARM}{\text{Maximize}} \quad \mathcal{F}(x) = \mathcal{F}_{\text{fitness}}(x) + \mathcal{F}_{\text{feasibility}}(x). \tag{5.1}$$

The function is expressed as the sum of the evaluation of the fitness of a particle, $\mathcal{F}_{\text{fitness}}(x)$, and the evaluation of the feasibility of a particle, $\mathcal{F}_{\text{feasibility}}(x)$. The fitness of a particle concerns how well the it performs on the objectives of the problem, whereas the feasibility of a particle addresses the degree to which the particle satisfies the constraints of the problem. These two evaluations, therefore, are intended to help guide the algorithm to generate (near-)optimal and valid solutions.

5.3.3.1 Objective functions

In this approach, two objective functions are used to assess the fitness of each particle of the swarm. These concern the duration of the project ($\mathcal{F}_{\text{duration}}(x)$) and the experience of the developers assigned ($\mathcal{F}_{\text{experience}}(x)$), as shown in Equation (5.2).

$$\mathcal{F}_{\text{fitness}}(x) = (w_1 \times \mathcal{F}_{\text{duration}}(x)) + (w_2 \times \mathcal{F}_{\text{experience}}(x)) \tag{5.2}$$

where $0 \leq w_i, w_2 \leq 1$ and $w_i + w_2 = 1$. Again, using a weighted-sum approach allows a software project manager to regulate the importance of each objective.

5.3.3.1.1 Project duration objective function

This first objective relates to the generation of (near-)optimal resource allocation and task scheduling solutions so that a software project completes within the shortest duration possible function. The approach uses the same objective function described by Equation (4.6), such that

$$\mathcal{F}_{\text{duration}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{delay}}. \tag{4.6 revisited}$$

The function aims to minimize the duration of a software project by calculating the fitness of each individual x in the swarm based on the degree to which unnecessary delays exist between tasks. As already mentioned, the function only requires the task scheduling information regarding the start time of each task.

5.3.3.1.2 Developer experience objective function

This objective works towards the generation of (near-)optimal resource allocation and task scheduling solutions in order that tasks are undertaken by the most experienced developers. The approach adopts the same objective function defined in Equation (4.9), where

$$\mathcal{F}_{\text{experience}}(x) = \frac{1}{m}\sum_{i=1}^{m} t_i^{\text{experience}}. \qquad \text{(4.9 revisited)}$$

The goal of the function is to maximize the experience of assigned developers by assessing the fitness of each individual x in the swarm based on the level of experience that developers possess in the various skills required by the tasks of the project. As stated previously, the function only makes use of the resource allocation information regarding which developers are assigned to each task.

5.3.3.2 Constraint functions

Several constraint functions are now introduced to handle the feasibility of solutions. They are aggregated together as shown in Equation (5.3):

$$\mathcal{G}_{\text{feasibility}}(x) = (cp_1 \times \mathcal{G}_{\text{dependencies}}(x)) + (cp_2 \times \mathcal{G}_{\text{skills}}(x)) + (cp_3 \times \mathcal{G}_{\text{assingments}}(x)). \qquad (5.3)$$

Each constraint is multiplied by a negative coefficient, $cp_1, cp_2, cp_3 < 0$, as a form of penalty for when each constraint is violated. This allows a software project manager the option of relaxing the importance of certain constraints while stressing the significance of others. Specifically, the constraints considered in this approach assess the degree to which: (1) dependencies are satisfied by the scheduling of tasks, (2) skills required by tasks are fulfilled by the allocation of resources, and (3) assignments comply with the availability of developers. Ideally, the value of $\mathcal{G}_{\text{feasibility}(x)}$ for a particle x should have a value of zero (the maximum possible) indicating that all constraints are satisfied. Otherwise, the function will take a negative value denoting that one or more constraints are not met. Hence, the lower the value, the less feasible the solution represented by the particle.

5.3.3.2.1 Dependency satisfaction constraint function

The first constraint addresses the generation of valid resource allocation and task scheduling solutions in a way which ensures that all task dependencies are satisfied as a result of the scheduling of each task. This constraint is defined by the dependency satisfaction constraint function, whose

goal is to evaluate the feasibility of each particle x in the swarm based on the number of instances where a task's start time is scheduled before the finish times of all its predecessors. Accordingly, the function only makes use of the task scheduling information of an individual.

Initially, the algorithm calculates the dependency violations, $t_i^{\text{dviolations}}$, for each task in the project. Specifically, in the case where task t_i is scheduled to begin before its earliest start time, the algorithm multiplies the number of time units that are violated by the number of successor tasks, suc_i, that depend on t_i using the following conditional formula

$$t_i^{\text{dviolations}} = \begin{cases} (t_i^{\text{earliest}} - t_i^{\text{start}}) \times suc_i, & t_i^{\text{start}} < t_i^{\text{earliest}} \\ 0, & \text{otherwise} \end{cases}. \tag{5.4}$$

The purpose of this multiplication is to help correct subsequent paths in the TPG that involve the violating task. Once the dependency violations of each tasks has been calculated, the total value for this constraint is determined by averaging the total number of dependency violations of all tasks using

$$\mathcal{G}_{\text{dependencies}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{dviolations}}. \tag{5.5}$$

So, the lower the number of unsatisfied dependencies per task, then the smaller the value of the constraint function $\mathcal{G}_{\text{dependencies}}(x)$, and ergo the higher the degree of feasibility of the solution represented by particle x.

5.3.3.2.2 Skill fulfilment constraint function

The approach's second constraint regards the generation of valid resource allocation and task scheduling solutions in a way which ensures that all skills required by a task are fulfilled based on the allocation of resources. This constraint is defined by the skill fulfilment constraint function, which aims to assess the feasibility of each particle x in the swarm based on the number of instances where none of the developers assigned to a task possesses experience in a particular skill. Thus, the function only requires of the resource allocation information of an individual.

To begin with, the conditional formula of Equation (5.6) is used to determine the skill violations, $t_i^{\text{sviolations}}$, of each task in the project. Here, the algorithm specifically calculates the average number of skills that are not fulfilled by the developers assigned to task t_i by using the value $treq_{ik}^{experience}$ calculated by the developer experience objective function (Equation (4.9)). In essence, the algorithm simply averages the total number of skills whose level of experience

$treq_{ik}^{\text{experience}}$ equals to zero, shown as

$$t_i^{\text{sviolations}} = \frac{1}{t_i^{\text{skills}}} \sum_{k=1}^{l} [treq_{ik} = 1 \wedge treq_{ik}^{\text{experience}} = 0], \tag{5.6}$$

where t_i^{skills} denotes the number of skills required by task t_i. If all skills can be fulfilled by the assigned developers then $t_i^{\text{sviolations}} = 0$. However, if none of the skills can be satisfied by the assigned developers then $t_i^{\text{sviolations}} = 1$. After the skill violations of each task has been calculated, the total value for this constraint is established by averaging the total number of skill violations of all tasks denoted in

$$\mathcal{G}_{\text{skills}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{sviolations}}. \tag{5.7}$$

Hence, the fewer the number of unfulfilled skills per task then the smaller the value of $\mathcal{G}_{\text{skills}}(x)$, and consequently the higher the degree of feasibility of the solution represented by the particle x.

5.3.3.2.3 Assignment validity constraint function

One of the issues observed in Approach 1 described in Chapter 4 was the fact that the availability of developers was incorrectly handled as an objective rather than as a constraint. If an individual in the swarm represents the case where a developer is assigned to more than one task at any given time in the project, then this should affect the feasibility of the solution, and not its optimality. The final constraint, therefore, relates to the generation of valid resource allocation and task scheduling solutions in a way which ensures that assignments comply with the availability of developers. This constraint is defined by the assignment validity constraint function, the purpose of which is to ascertain the feasibility of each particle x in the swarm based on the number of instances where a developer has been assigned to work on more than one task simultaneously. The function therefore utilizes both the resource allocation and task scheduling information of an individual in a slightly modified version of the assignment validity objective function described in Paragraph 4.3.3.1.3 in Approach 1.

First, the algorithm calculates the number of time units a developer r_j has been assigned to more than one task, $r_j^{\text{conflicts}}$, using the information regarding at which time and on which task the developer is to work on, and then divides this number by the number of time units that the developer has been assigned to work in total, $r_j^{\text{totassigned}}$, as shown in

$$r_j^{\text{aviolations}} = \frac{r_j^{\text{conflicts}}}{r_j^{\text{totassigned}}}. \tag{5.8}$$

After the assignment violations of each developer has been computed, the total value for this constraint is calculated by averaging the total number of assignment violations of all developers by applying

$$\mathcal{G}_{\text{assignments}}(x) = \sum_{j=1}^{n} r_j^{\text{aviolations}}. \tag{5.9}$$

Consequently, the fewer the noncompliant assignments per developer then the smaller the value of $\mathcal{G}_{\text{assignments}}(x)$, and therefore the higher the degree of feasibility of the solution represented by particle x.

5.4 Experiments

An experiment was carried out to assist in answering the research questions posed in Section 5.1 regarding the algorithm's ability to generate feasible and (near-)optimal solutions taking into account the assumptions and criteria put forward in the approach.

Initially, a small survey was conducted in order to find out the driving factors influencing the size and complexity of a software project. Specifically, three project managers working at software development SMEs in Cyprus were interviewed so as to ascertain the main features they take into account when performing task scheduling and developer allocation activities in software projects. The interviews consisted of open questions to allow the software project managers to express in their own words how they define the size and complexity of small, medium and large software projects. This procedure identified that project managers consider the number of the tasks and the number of skills required by each task as the primary features for the size of a software project. On the other hand, the complexity of the project is fundamentally specified by the number of dependencies between the tasks. In particular, project managers consider the dependency rate, which is defined as the ratio of the number of dependencies present in the project to the total number of possible dependencies in the project. Additionally, the number of developers and the number of skills they possess are also regarded as important features affecting the size of a software project and also its complexity.

Using the information provided by the project managers, a total of seven software project instances of varying size and complexity were generated aiming to represent real-world software project case studies, summarized in Table 5.1. The project instances were constructed in a way so as to allow, on the one hand, the evaluation of the performance of the algorithm in terms of optimality and, on the other hand, assessment of the performance of the algorithm in terms feasibility.

Table 5.1: Size and complexity of software projects used for experiments in Approach 2

Project instance	No. tasks	No. dependencies	Dependency rate	Avg. no. skills per task	No. available developers	Avg. no. skills per developer
P2A	10	13	29%	2.0	10	2.0
P2B	14	16	18%	2.0	10	1.5
P2C	18	24	16%	2.0	10	1.2
P2D	18	24	16%	2.0	5	0.7
P2E	25	15	5%	2.5	8	1.0
P2F	30	62	14%	3.3	18	2.0
P2G	30	62	14%	3.3	10	1.0

Specifically, the difficulty level between projects $P2A$–$P2C$ increases with respect to the project size and the complexity based on the number of tasks and the number of dependencies between each task, respectively. The difference in project $P2C$ and project $P2D$ lies in the number of developers and the average number of skills each developer possesses, influencing both complexity and size features making the latter project tougher to handle. The difficulty level increases even more in project $P2E$ where, although the dependencies rate is only 5%, the average number of required skills increases while the number of available developers decreases, compared to the first three projects. This represents the case where the project is marginally outside of the capacity of a software development company, since more skills are required but fewer developers are available. The last two projects ($P2F$ and $P2G$) represent more challenging tests, since the size and complexity of the projects are affected by all the driving factors. Among them, project $P2G$ is the most competitive because both the number of developers and the average skills per developer decrease.

Taking into account the fact that the project duration and developer experience objectives are considered to be directly competing, the PSO algorithm was executed to generate solutions for the seven project instances with three different weight preferences. The preferences of the two objectives were adjusted by modifying the weights w_1 and w_2 shown in Equation (5.2). The purpose of this was to monitor how the change influences the evolutionary behaviour of the algorithm and the quality of the final solutions generated. In the first weight preference, both objectives are given equal importance, thus setting their weights $w_1 = w_2 = 0.5$. In the second weight preference, the project duration is considered more significant and, therefore, the weights were set in favour of the first objective function of Equation (5.2) with $w_1 = 0.9$ and $w_2 = 0.1$. Finally in the third

preference, team experience was considered more important and so the weights were set in favour of the second objective function of Equation (5.2) with $w_1 = 0.1$ and $w_2 = 0.9$.

Among the numerous variations of the particle swarm optimization algorithm, Binary-PSO [86] and Constriction-PSO [87] were selected as the most suitable and promising implementations that would be able to handle the task scheduling and developer allocation problem. The former is used to control the explosion of the swarm and to improve the convergence of particles over time, whereas the latter is adopted particularly to handle developer allocation due to the particles' bitset representation of assigned developers. Furthermore, the multimodal nature of the problem, having many global/local minimum, dictates the utilization of a low-connected topology in order for the swarm to adequately examine the search space and to avoid premature convergence in local (near-)optimal solutions. Due to this, the ring topology was used, in which a particle is connected with its left- and right-adjacent neighbours. Additionally, the swarm size during the experimental procedure was kept constant at 60 particles. In case that stagnation was observed, that is, if for a certain percentage of iterations there was no improvement in the best globally found solution, a partial reinitialization of positions and velocities took place so as to remove stagnation and give an additional boost to the particles. Finally, the value of the penalty weights for the constraints were all specified to $cp_1 = cp_2 = cp_3 = -100$ giving, thus, equal importance to the constraints.

Each project instance was run ten times for each of the three weight preferences, with a maximum number of iterations set to 10^6. The resource allocation and task schedule solutions obtained at the end of each run for all weight preferences were then analyzed to help answer research question RQ2.1 regarding the feasibility of the generated solutions and research question RQ2.2 regarding the optimality of the generated solutions.

5.5 Results and discussion

5.5.1 RQ2.1: How well do the constraint functions perform in helping the algorithm generate feasible resource allocation and task scheduling solutions?

In order to answer the first research question (RQ2.1), each particle in the swarm was assessed based on whether it represented a feasible resource allocation and task scheduling solution. For this purpose, a swarm feasibility rate was calculated at the end of each run for each preference separately, defined as the percentage of solutions in the swarm that do not violate any of the three constraints analysed in Section 5.3.3.2. Then, the average feasibility rate was computed over all

ten runs of the algorithm per objective weight preference. The results of this experiment are shown in Table 5.2.

Table 5.2: Average feasibility rate of generated solutions for each project instance using three different objective function weight preferences in Approach 2

	Project instance						
Weight preference	$P2A$	$P2B$	$P2C$	$P2D$	$P2E$	$P2F$	$P2G$
$w_1 = 0.5, w_2 = 0.5$	100.0	99.1	97.9	96.6	89.0	88.2	85.0
$w_1 = 0.9, w_2 = 0.1$	100.0	99.6	97.8	95.6	88.8	87.3	83.8
$w_1 = 0.1, w_2 = 0.9$	100.0	98.0	96.0	95.0	90.0	89.0	87.0

As can been seen from the results, the particles at the end of the runs for project $P2A$ all represent feasible solutions for each different preference. As the complexity and size of the software projects increase however, the feasibility ratios begin to decrease. Despite this fall, the majority of the solutions that the algorithm generates are always feasible (but not necessarily optimal) even in the most complex and difficult scenarios (projects $P2E$–$P2G$). This indicates that the algorithm is highly capable in constructing sufficient solutions with respect to the hard constraints imposed in this approach. An important observation is the similar rate of decrease in the feasibility of the solutions for all three objective weight preferences from 100% in project $P2A$ to around 84–87% in project $P2G$. This is basically due to the progressive increase in complexity and difficulty levels from project to project. Thus, solving the optimization problem becomes more complicated and challenging, especially with the multimodal nature of the problem containing many local and global optima, which causes the the algorithm to struggle to produce feasible solutions.

Another noteworthy observation that was made after examining particles representing infeasible solutions, is that the only constraint not satisfied was that of developer availability (Equation (5.9)). No dependency violations were recorded and also the skills required by each task were all fulfilled by the developers assigned. One possible explanation for this behaviour can be attributed to the actual objectives of the problem. The constraint responsible for avoiding conflicts in developer availability is influenced by both objectives and, therefore, it is more difficult to be satisfied, whereas the other two constraints are impacted only by one of the objective functions –- the former by project duration and the latter by developer experience.

5.5.2 RQ2.2: How well do the objective functions perform in helping the algorithm generate (near-)optimal resource allocation and task scheduling solutions?

The purpose of the second experiment was to help answer research question RQ2.2., and focused on examining the algorithm with respect to its ability to generate (near-)optimal solutions, that is, solutions where no unnecessary delays exist in the task scheduling and each task is allocated the most experienced developer(s). The evaluation uses the hit rate of a swarm, defined as the percentage of particles in the swarm that represent (near-)optimal solutions at the end of each run. Because the project instances were optimized ten times for each weight preference, the average hit rate was computed for each project instance for all three weight preferences individually. The results are provided in Table 5.3.

Table 5.3: Average hit rate of generated solutions for each project instance using three different objective function weight preferences in Approach 2

	Project instance						
Weight preference	$P2A$	$P2B$	$P2C$	$P2D$	$P2E$	$P2F$	$P2G$
$w_1 = 0.5, w_2 = 0.5$	100.0	30.0	50.0	30.0	0.0	0.0	0.0
$w_1 = 0.9, w_2 = 0.1$	100.0	50.0	40.0	30.0	0.0	0.0	0.0
$w_1 = 0.1, w_2 = 0.9$	100.0	90.0	80.0	70.0	0.0	0.0	0.0

Studying the hit rates in Table 5.3, it can be seen that the algorithm performs sufficiently well in the first four projects in all three weight preferences. In these projects, the hit rates reach a maximum value of 100% for project $P2A$ (the simplest of all project instances) but decrease progressively (reaching as low as 30% in project $P2D$, as the software projects become more difficult. As mentioned previously, the feasibility rate decreases approximately the same way for all experiments in the first four projects, but this does not occur in the case of the hit rates, where fluctuations can be seen.

Comparing results for projects $P2C$ and $P2D$, there is an obvious decrease in the number of (near-)optimal solutions produced. This could be attributed to the fact that even though the number of tasks, dependencies and required skills are the same, there are fewer developers available to carry out project $P2D$ than project $P2C$, who also possess a lower number of skills on average. A possible explanation for the behaviour of the algorithm in project $P2D$, therefore, is that the algorithm encounters more difficulties when trying to satisfy the constraints since, intuitively, the fewer the number of available developers, the more likely that conflicts in developer availability will arise. Thus, the algorithm produces solutions with lower quality (by creating idle gaps between

tasks) when compared to project $P2C$, despite the number of feasible solutions being, on average, roughly equal.

With regards to projects $P2E$ and $P2F$, the algorithm experiences some difficulties in finding (near-)optimal solutions, despite being able to frequently generate feasible solutions (within 80–90% of the time). This can suggest that the increase in the complexity and size of software projects causes difficulties in the evolution of the algorithm and, consequently, to the generation of acceptable solutions. For example, in project $P2E$ the rate of dependencies is low (only 5%), which implies that many tasks can start executing simultaneously. In fact, for this project 13 tasks can start executing on the first day; hence making the scheduling of tasks by the algorithm much harder as it tries to satisfy all three constraints simultaneously. Furthermore, the imbalance between the average number of skills required by each task and the skills possessed by each developer raises the level of difficulty to such a degree that precise adaptations and movements of task starting days are required by the algorithm in order to convert feasible solutions to (near-)optimal solutions as well. These issues also apply in the case of projects $P2F$ and $P2G$. However, better results were achieved in the project $P2E$, since fewer unnecessary gaps were recorded in the results.

Another significant observation relates to the variety of (near-)optimal solutions generated by the algorithm (that is, the uniqueness of the resource allocations and task schedules). As previously mentioned, each weight preference was run ten times for all projects, meaning a total of 30 executions of the algorithm per project. The number of different (unique) solutions identified for projects $P2A$–$P2D$ per weight preference is shown in Table 5.4. Careful examination of these solutions actually showed that the solutions differed only with respect to resource allocation, while their task schedules remained exactly the same.

Table 5.4: Number of unique solutions generated for projects $P2A$–$P2D$ using three different objective function weight preferences in Approach 2

	Project instance			
Weight preference	$\boldsymbol{P2A}$	$\boldsymbol{P2B}$	$\boldsymbol{P2C}$	$\boldsymbol{P2D}$
$\boldsymbol{w_1 = 0.5, w_2 = 0.5}$	4	2	1	1
$\boldsymbol{w_1 = 0.9, w_2 = 0.1}$	5	2	2	4
$\boldsymbol{w_1 = 0.1, w_2 = 0.9}$	4	2	4	3

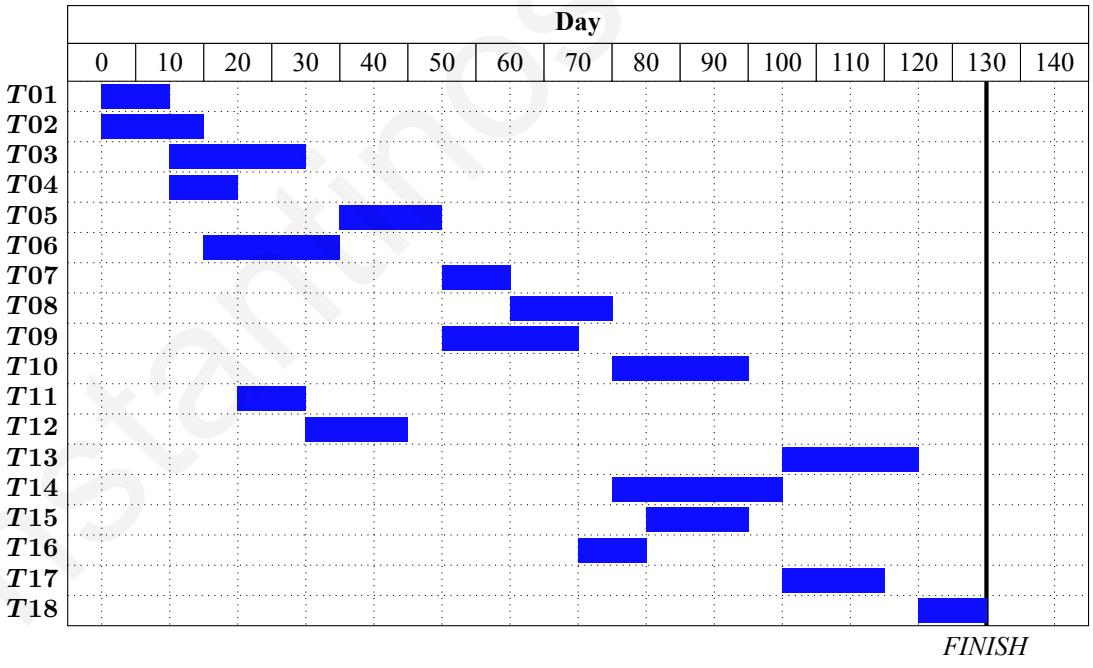
The objective weights included in the evaluation of a swarm's particles allows the algorithm to be guided into generating solutions that favour either one of the two objective functions. Again using projects $P2C$ and $P2D$ as a comparison, the objective function values of the overall best

solutions for the different experiments (Table 5.5) present the visible trade-off and competitive nature of the objectives.

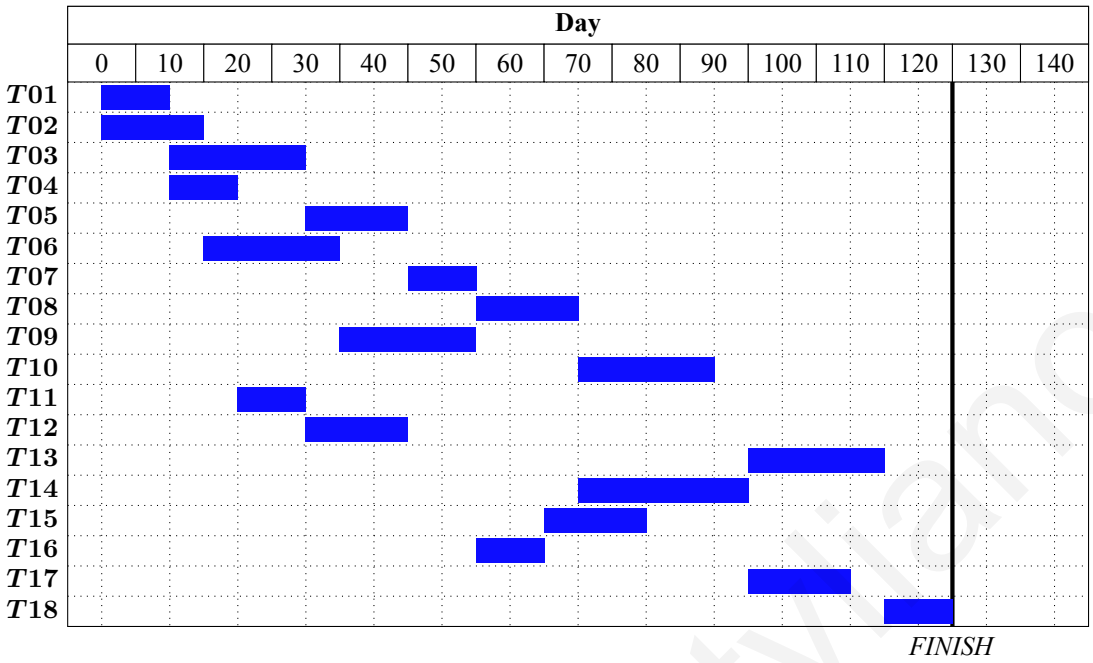
Table 5.5: Objective function values obtained for project $P2C$ and $P2D$ using three different objective function weight preferences in Approach 2

	$P2C$		$P2D$	
Weight preference	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$
$w_1 = 0.5, w_2 = 0.5$	0.984	0.609	0.979	0.608
$w_1 = 0.9, w_2 = 0.1$	0.991	0.605	0.982	0.604
$w_1 = 0.1, w_2 = 0.9$	0.971	0.612	0.966	0.612

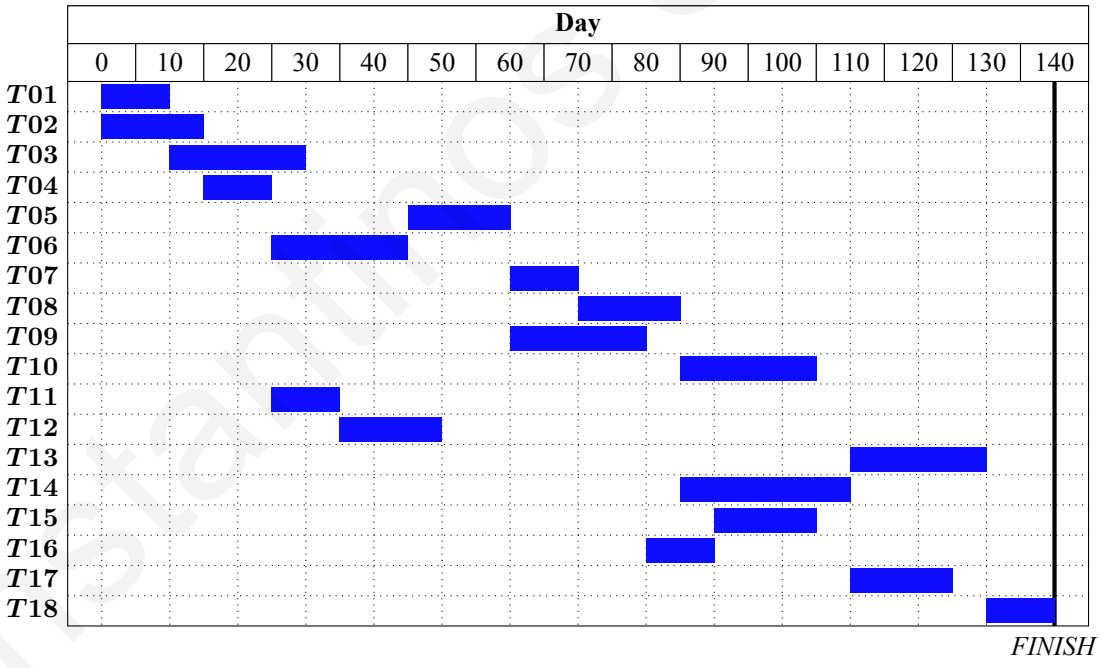
With greater emphasis on developer experience, the algorithm yields its highest fitness value for $\mathcal{F}_{\text{experience}}$ and its lowest for $\mathcal{F}_{\text{duration}}$. Conversely, with greater significance on project duration, the algorithm produces its highest fitness value for $\mathcal{F}_{\text{duration}}$ and its lowest for $\mathcal{F}_{\text{experience}}$. When equal importance is given, both the objective function values of the best solution are somewhere between the best and worse for each objective. This can also be seen in Figure 5.1, which presents the Gantt charts of the best solutions generated for project $P2C$ with each of the three weight preferences.



(a) Total project duration of the best solution found for project $P2C$ with weight preference $w_1 = w_2 = 0.5$ is 130 days



(b) Total project duration of the best solution found for project $P2C$ with weight preference $w_1 = 0.9$ and $w_2 = 0.1$ is 125 days



(c) Total project duration of the best solution found for project $P2C$ with weight preference $w_1 = 0.1$ and $w_2 = 0.9$ is 140 days

Figure 5.1: Gantt charts of the best solutions found for project $P2C$ using three different objective function weight preferences in Approach 2

As can be seen in the figure, the different assignments of developers produce varying project durations due to the distribution of the objective function weights. With an equal preference in weights, the total duration of the project is 130 days, which is longer than the duration of the project generated with a higher preference for project duration (125 days), but shorter than the duration of the project generated with a higher preference for developer experience (140 days).

5.6 Summary

The results obtained from various executions of the algorithm indicated that particle swarm optimization is a promising approach for software task scheduling and developer allocation, and performs sufficiently well in the majority of the test projects examined. The average feasibility rate of the solutions generated is more than 83%, proving that most of the particles in a swarm reside in feasible search space area. However, some difficulties were encountered in scenarios consisting of larger sized and more complex software projects, where the number of tasks, the rate of dependencies and the number of available developers were shown to influence the ability of the algorithm to produce (near-)optimal solutions. By observing the solutions generated, several issues were encountered. First, it was observed that some solutions contained an overallocation of experienced developers to certain tasks. Although there were no noncontributing developers assigned to these tasks, the algorithm tended to select too many experienced developers, when in reality the solutions would still be as good even with the experience of fewer developers. Second, and most important issue, is that needless gaps between tasks continued to exist just as in the previous approach, despite solutions satisfying all constraints imposed, which is the main reason that the average hit rates in all three weight preferences showed a decrease. This led to the creation of the approach described next in Chapter 6, similar to the previous two, but with enhancements to mitigate the previous issues.

Chapter 6

Approach 3: Optimizing Project Duration and Developer Experience using a Pareto Ranking Genetic Algorithm

The main purpose of this approach [88] is to modify the optimization function described in the previous two proposed approaches in an attempt to improve the quality of solutions generated, particularly regarding the project duration objective, since the phenomenon of unnecessary delays between tasks remained. Furthermore, the two approaches presented up until now both use a weighted-sum approach, which essentially combines the objectives functions into a scalar function with one objective. However, the main drawback with this method is that a project manager is required to select an appropriate weight for each objective function beforehand, which is not always ideal or reflective of the importance of each objective. In addition, because the objectives can be conflicting, it would be more beneficial for project managers to have a set of (near-)optimal solutions consisting of the possible trade-offs between the objectives to chose from after the optimization.

In order to overcome these issues, the approach uses the Non-dominated Sorting Genetic Algorithm (NSGA-II) [89], which is a genetic algorithm that applies the concept of Pareto ranking on individuals in order to generate a set of (near-)optimal solutions in multi-objective environments. For comparison, the approach also uses the Archived Multi-objective Simulated Annealing (AMOSA) algorithm [90], which is a search-based technique again used to solve multi-objective optimization problems.

6.1 Research questions

The primary research target is to compare the performance of the two multi-objective algorithms. Specifically, research question (RQ3.1) aims to compare the quality of the solutions generated by two algorithms considering the objectives and constraints underlying this modified approach: **RQ3.1: How do NSGA-II and AMOSA perform with respect to generating feasible and (near-)optimal resource allocation and task scheduling solutions?**

6.2 Problem description

This third approach borrows the notations provided in Section 4.2, which were used to define the resource-constrained software project scheduling problem put forward in the two previous approaches proposed in Chapters 4 and 5 .

6.3 Pareto ranking genetic algorithm method

The purpose of using a Pareto ranking genetic algorithm method is to generate a set of (near-)optimal resource allocation and task scheduling solutions (rather than just one single (near-)optimal solution) based the multiple assumptions and objectives defined in the approach. This set of (near-)optimal solutions is known as the Pareto optimal set, and contains only those solutions that are nondominated by (or noninferior to) others in the set. In other words, each (near-)optimal solution represents a particular trade-off between the objectives, where any improvement in one of the objectives leads to the worsening of one or more other objectives. Because this approach works as an a posteriori method, that is, the decision-maker does not provide any preferences regarding the importance of each objective before the optimization, the decision-maker is free to adopt any one of the (near-)optimal resource allocation and task scheduling solutions generated [91].

The Pareto ranking genetic algorithm adopted in this approach is the second version of the Non-dominated Sorting Genetic Algorithm (NSGA-II) proposed by Deb et al. in 2002 [89]. The general outline of the algorithm along with the complementary notations can be found in Algorithm A.3. Since there are constraints present in the task scheduling and developer allocation (that is, solutions may be infeasible because of dependency relationship violations and/or assignment conflicts) the constrained NSGA-II algorithm was implemented. One important characteristic of NSGA-II is that it promotes the diversity of solutions through a crowded comparison operator used

both during the tournament selection and population reduction steps of the algorithm. Also, elitism is guaranteed since parent and offspring populations are combined prior to being sorted according to nondomination.

In addition, a multi-objective simulated annealing algorithm was also implemented using the Archived Multi-Objective Simulated Annealing (AMOSA) algorithm proposed by Bandyopadhyay et al. [90]. Simulated annealing mimics the process of physically heating materials for a certain amount of time at different temperature intervals, starting at a high temperature and slowly lowering the temperature over time to reduce defects [92]. This specific variation stores nondominated solutions in an archive that is updated iteratively based on the comparison of the suitability and feasibility of a current solution and a newly created solution (resulting from the mutation of the current solution). The same objective and constraints functions are incorporated into the implementation to allow direct comparison of the solutions in the final populations with respect to the feasibility and optimality of the task schedules and developer allocations represented by individuals. The steps of the algorithm are outlined in Algorithm A.4.

6.3.1 Representation and encoding

In order to generated (near-)optimal resource allocation and task scheduling solutions, the algorithm requires the same two pieces of information described in Section 4.3.1. Each solution is represented by an individual made up of a mixed-type array in which each element, similarly to before, contains the information regarding the starting time of each task (encoded using a nonzero positive integer) and the assignment of developers (encoded using a bitset).

6.3.2 Population initialization

Initialization of individuals in the population follows the same procedure described in Section 4.3.2. Again, the main goal is to start with a generation of feasible solutions, and therefore no individual is allowed to have any tasks without developers able to carry it out.

6.3.3 Optimization function

A new optimization function is implemented in this approach so as to allow the Pareto ranking method to find a set of (near-)optimal solutions that schedule task with the shortest possible duration and allocate developers with the highest experience. Specifically, the function aims to

maximize a vector of three objective functions subject to three constraints, as shown in Equation (6.1):

$$\begin{aligned} \underset{\forall x \in POP}{\text{Maximize}} \quad & \mathcal{F}(x) = (\mathcal{F}_{\text{duration}}(x), \mathcal{F}_{\text{experience}}(x), \mathcal{F}_{\text{size}}(x)), \\ \text{subject to} \quad & \mathcal{G}_{\text{dependency}}(x) = 0, \\ & \mathcal{G}_{\text{skills}}(x) = 0, \\ & \mathcal{G}_{\text{assignment}}(x) = 0. \end{aligned} \tag{6.1}$$

6.3.3.1 Objective functions

The first two objective functions, $\mathcal{F}_{\text{duration}}(x)$ and $\mathcal{F}_{\text{experience}}(x)$, are adopted as-is from the previous two proposed approaches described in Chapters 4 and 5. The third objective function, $\mathcal{F}_{\text{size}}(x)$, attempts to minimize the number of developers assigned to each task in order to handle the issue of over-allocating developers to tasks.

6.3.3.1.1 Project duration objective function

This objective relates to generating resource allocation and task schedule solutions so that a software project completes within the shortest duration possible. The approach uses the same objective function described in Equation (4.6), defined by

$$\mathcal{F}_{\text{duration}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{delay}}. \tag{4.6 revisited}$$

The function's purpose is to minimize the duration of a software project by calculating the fitness of each individual x in the population taking into account the degree to which unnecessary delays exist between tasks.

6.3.3.1.2 Developer experience objective function

This objective concerns generating resource allocation and task schedule solutions in a way where tasks are undertaken by the most experienced developers. The approach adopts the same objective function from Equation (4.9), such that

$$\mathcal{F}_{\text{experience}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{experience}}. \tag{4.9 revisited}$$

The function targets the maximization of the experience of assigned developers by assessing the fitness of each individual, x, in the population based on the level of experience that developers possess in the various skills required by the tasks of the project.

6.3.3.1.3 Team size objective function

The final objective in this approach relates to generating resource allocation and task schedule solutions in a way where the over-allocation of resources is reduced. Fewer team members mean less cost and effort devoted to a project and less communication overhead, the latter being something that software project managers are aware of and almost always want to avoid. More importantly, it allows for more developers to be available for selection in other tasks. This objective is expressed in the project team size objective function, which aims to minimize the size of a team by assessing the fitness of individuals in the population based on the number of developers assigned to each task. Therefore, the function only requires the resource allocation information regarding which developers are assigned to each task.

To begin with, for each task t_i in the project, the algorithm calculates t_i^{size} by taking into account the number of developers, t_i^{assigned}, assigned to task t_i using

$$t_i^{\text{size}} = \frac{1}{1 + t_i^{\text{assigned}}}. \tag{6.2}$$

Next, the total sum of the team sizes calculated using Equation (6.2) are averaged over the number of tasks with

$$\mathcal{F}_{\text{size}}(x) = \frac{1}{m} \sum_{i=1}^{m} t_i^{\text{size}}. \tag{6.3}$$

The higher the partial values of t_i^{size}, then the fitter the individual with regards this objective due to the overall number of developers assigned to work on each task being lower.

6.3.3.2 Constraint functions

The three constraint functions used to handle the assumptions in the approach remain the same as those described in Section 5.3.3.2. In particular, the approach employs these functions in order to guide the algorithm to generate feasible solutions where: (1) dependencies are satisfied by the scheduling of tasks (dependency satisfaction constraint function), (2) skills required by tasks are fulfilled by the allocation of resources (skill fulfilment constraint function), and (3) assignments comply with the availability of developers (assignment validity constraint function).

6.3.3.2.1 Dependency satisfaction constraint function

The dependency satisfaction constraint, $\mathcal{G}_{\text{dependencies}}(x)$, assesses the feasibility of each individual x in the population based on the degree to which task dependencies are satisfied as a result of

the scheduling of each task. The value of the function is calculated using Equation (5.5) described in Paragraph 6.3.3.2.1 in Approach 2.

$$\mathcal{G}_{\text{dependencies}}(x) = \frac{1}{m}\sum_{i=1}^{m} t_i^{\text{dviolations}}. \qquad \text{(5.5 revisited)}$$

6.3.3.2.2 Skill fulfilment constraint function

The skill fulfilment constraint, $\mathcal{G}_{\text{skills}}(x)$, assesses the feasibility of each individual x in the population based on the degree to which skills required by a task are fulfilled based on the allocation of resources. The value of the function is calculated using Equation (5.7) described in Paragraph 5.3.3.2.2 in Approach 2.

$$\mathcal{G}_{\text{skills}}(x) = \frac{1}{m}\sum_{i=1}^{m} t_i^{\text{sviolations}}. \qquad \text{(5.7 revisited)}$$

6.3.3.2.3 Assignment validity constraint function

The assignment validity, $\mathcal{G}_{\text{assignments}}(x)$, assesses the feasibility of each individual x in the population based on the degree to which assignments comply with the availability of developers. The value of the function is calculated using Equation (5.9) described in Paragraph 5.3.3.2.3 in Approach 2.

$$\mathcal{G}_{\text{assignments}}(x) = \sum_{j=1}^{n} r_j^{\text{aviolations}}. \qquad \text{(5.9 revisited)}$$

6.4 Experiments

An experiment was carried out to help answer research question RQ3.1 regarding the quality of the resource allocation and task scheduling solutions generated by NSGA-II and AMOSA, subject to the approach's objectives and constraints.

A total of eight software project instances with varying size and complexity were again created after consulting with three expert project managers with the purpose of representing possible instances of real-world software projects. The software project managers were interviewed with open questions to extract the features of these software projects, which are presented in Table 6.1.

Project $P3A$ provides an example of a project to be undertaken by a small software company with few available developers each possessing a moderate number of skills. The project contains very few tasks and each task requires only one skill. In addition, the project contains a small number of dependencies and so is considered to have low complexity. Project $P3B$ again involves

Table 6.1: Size and complexity of software projects used for experiments in Approach 3

Project instance	No. tasks	No. dependencies	Dependency rate	Avg. no. skills per task	No. available developers	Avg. no. skills per developer
P3A	5	5	50%	1.0	3	3.0
P3B	12	26	39%	1.0	3	1.0
P3C	15	40	38%	2.0	8	2.0
P3D	18	36	24%	4.0	10	3.0
P3E	20	52	27%	2.0	3	2.0
P3F	30	62	14%	3.0	18	3.0
P3G	20	65	34%	3.0	6	4.0
P3H	25	16	5%	3.0	3	3.0

a small-sized company undertaking a software project, but this time with both a moderate number of tasks and dependencies. As in project $P3A$, each task requires only one skill and, furthermore, developers in the company specialize also in only one skill. Projects $P3C$ and $P3D$ both represent examples of projects to be performed by medium-sized software companies. However, in project $P3D$ the size of the project is larger but there are fewer dependencies between tasks. Another difference is that tasks in project $P3D$ require more skills and developers possess experience in a wider range of skills.

Projects $P3E$ and $P3F$ both correspond to a software project that is both large in size and high in complexity. In the case of project $P3E$, the project is to be undertaken by a small development company with developers that possess a moderate range of skills, whereas in project $P3F$ a larger software company will be responsible for the project's development with its developers having experience in a larger number of skills on average. Also, tasks in project $P3E$ require, on average, fewer skills each. Project $P3G$ in the experiments consists of a large and more complicated project undertaken by a medium-sized company. This project is similar to project $P3E$, however, with two differences: each task requires more skills and developers possess experience in a larger number of skills. Finally, project $P3H$ represents a large but less complex project with tasks requiring a medium number of skills. The size of the development company is small but developers have experience in an average number of different skills.

Both NSGA-II and AMOSA were run 15 times for each project with 3500 iterations per run. All runs of the NSGAI-II consisted of a population with 100 individuals. The objective functions and constraint functions were applied to each solution and the evolution process began by selecting a number of parents to enter the mating pool using a tournament selection procedure of size four.

The two best parents of each tournament in terms of fitness and/or feasibility were selected for recombination and mutation. The operators were applied uniformly across each variable so that each project task had a probability of being recombined and mutated with a value of 0.90 and $2/m$, respectively. In addition, simulated binary crossover for the integer-coded variables was used for reproduction of variables representing start days, while single-point binary crossover was adopted for reproduction of the bitset-encoded variables representing developer assignments. Mutation of the integer-coded variables used polynomial mutation, whereas the bitset-coded variables used a bit-flip operator.

6.5 Results and discussion

6.5.1 RQ3.1: How do NSGA-II and AMOSA perform with respect to generating feasible and (near-)optimal resource allocation and task scheduling solutions?

The first property examined in the comparison was the number of unique solutions present in the final Pareto front (NSGA-II) and final Pareto archive (AMOSA) of each run. Table 6.2 summarizes this data for all eight projects, for both the NSGA-II and AMOSA implementations of the proposed approach. For project $P3A$, NSGA-II runs generated final Pareto fronts consisting of either one, two or three unique solutions, indicating that the majority of individuals in a population converged to the same point(s) in the search space. However, this range in number seems to increase as the attributes of the software projects increase, both in size and complexity.

Also for each project, the final Pareto fronts/archives of the 15 runs were pooled together into one approximation Pareto front/archive in order to identify the combined set of (near-)optimal solutions as a whole for each algorithm. The number of best solutions in these fronts/archives for each project are also given in Table 6.2. For example, AMOSA generated a total of 82 unique solutions for project $P3A$ (summation of the column) over the 15 iterations, averaging around six unique solutions per run. But after combining the 15 final Pareto archives, a total of three (near-)optimal solutions were found to comprise the approximation Pareto archive.

Table 6.2: Results of obtained by NSGA-II and AMOSA in the experiments performed in Approach 3

	Project $P3A$		Project $P3B$		Project $P3C$		Project $P3D$	
	NSGA-II	**AMOSA**	**NSGA-II**	**AMOSA**	**NSGA-II**	**AMOSA**	**NSGA-II**	**AMOSA**
Run 01	1	6	2	13	100*	100*	98	100*
Run 02	3	2	1	53	100*	81*	98	100*
Run 03	1	6	1	58	92	93*	99	100*
Run 04	3	6	7	39	60	97*	99	100*
Run 05	3	5	37	11*	100*	100*	95	100*
Run 06	1	10	96	44	100*	83*	97	100*
Run 07	2	11	1	35	3	88*	95	100*
Run 08	1	4	3	41	100*	72	98	100*
Run 09	2	10	1	35	73	100*	99*	100*
Run 10	2	6	70	14	100*	100*	97	100*
Run 11	2	6	43	21	100*	74*	97	100*
Run 12	2	7	5	2	100*	100*	97	100*
Run 13	1	1*	1	32	75*	65	100	100*
Run 14	2	5	6	34*	100*	75*	98	100*
Run 15	2	1	42	27	80	64*	95	100*
Total no. unique solutions	**28**	**86**	**316**	**459**	**1283**	**1292**	**1462**	**1500**
Avg. no. unique solutions	**2**	**6**	**21**	**31**	**86**	**86**	**97**	**100**
No. best solutions in approximation Pareto front	**2**	**3**	**5**	**4**	**8**	**2**	**215**	**1**

Table 6.2: Results of obtained by NSGA-II and AMOSA in the experiments performed in Approach 3 (continued from previous page)

	Project $P3E$		Project $P3F$		Project $P3G$		Project $P3H$	
	NSGA-II	**AMOSA**	**NSGA-II**	**AMOSA**	**NSGA-II**	**AMOSA**	**NSGA-II**	**AMOSA**
Run 01	99*	94*	100*	100*	100*	100*	30	100*
Run 02	99*	75*	100*	100*	100*	81*	98	100*
Run 03	100*	100*	94*	100*	80*	100*	89	100*
Run 04	98*	69*	9	100*	97*	100*	75	100*
Run 05	99*	100*	83	51*	100*	100*	64	100*
Run 06	98*	100*	100*	100*	100*	100*	75	100*
Run 07	6	95*	100*	88*	80	100*	99	100*
Run 08	99*	68*	100*	78*	100*	100*	81	100*
Run 09	100*	86*	100*	96*	99*	97*	100	100*
Run 10	99*	48*	98*	88*	90	100*	99	100*
Run 11	98*	92*	100*	100*	88	94*	72	100*
Run 12	100*	100*	100*	78*	83*	100*	100*	100*
Run 13	99*	100*	100*	93*	100*	100*	100	100*
Run 14	100*	100*	100*	100*	100*	100*	76	100*
Run 15	96*	100*	99*	100*	100*	100*	74	100*
Total no. unique solutions	**1390**	**1327**	**1383**	**1372**	**1417**	**1491**	**1232**	**1500**
Avg. no. unique solutions	**93**	**88**	**98**	**91**	**94**	**99**	**82**	**100**
No. best solutions in approximation Pareto front	**6**	**1**	**50**	**1**	**143**	**1**	**9**	**1**

Project $P3A$ involves a software project that is relatively small in size and low in complexity. The skills required by the project's tasks, as well as the experience that the available developers possessed in the skills the are shown in Table 6.3 and Table 6.4, respectively.

Table 6.3: Skills required for the tasks of project $P3A$ in Approach 3

	Required skill				
Task	$\boldsymbol{S01}$	$\boldsymbol{S02}$	$\boldsymbol{S03}$	$\boldsymbol{S04}$	$\boldsymbol{S05}$
$\boldsymbol{T01}$	✓				
$\boldsymbol{T02}$		✓			
$\boldsymbol{T03}$			✓		
$\boldsymbol{T04}$				✓	
$\boldsymbol{T05}$					✓

Table 6.4: Levels of experience of available developers in skills required for project $P3A$ in Approach 3

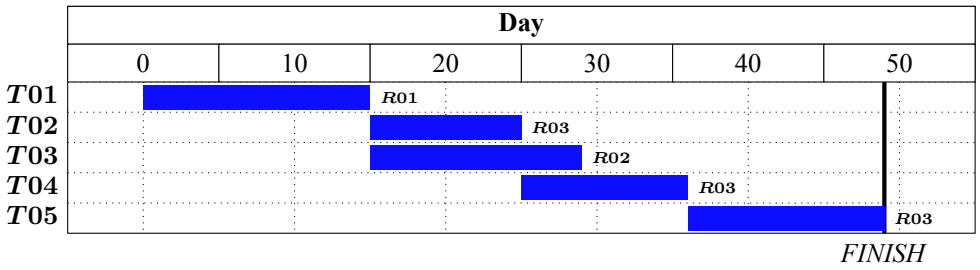
	Required skill				
Developer	$\boldsymbol{S01}$	$\boldsymbol{S02}$	$\boldsymbol{S03}$	$\boldsymbol{S04}$	$\boldsymbol{S05}$
$\boldsymbol{R01}$	0.88	0.00	0.00	0.01	0.15
$\boldsymbol{R02}$	0.00	0.82	0.74	0.29	0.00
$\boldsymbol{R03}$	0.00	0.36	0.14	0.56	0.63

Given the attributes of project $P3A$, it is possible to identify that there are two unique resource allocation and task schedule solutions ($3AS1$ and $3AS2$) for the project. The objective function values of these two optimal solutions are given in Table 6.5.

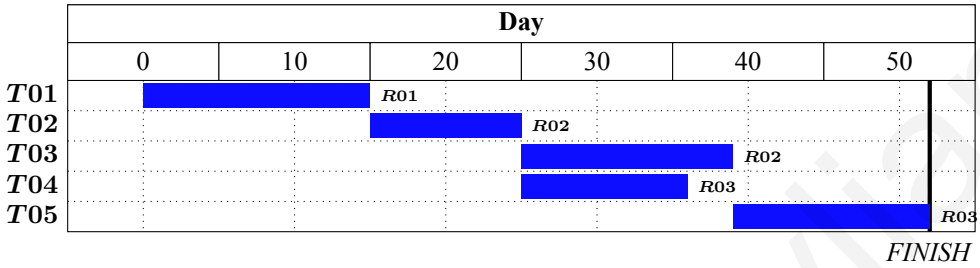
Table 6.5: Objective function values corresponding to the optimal solutions found by NSGA-II for project $P3A$ in Approach 3

	Objective function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\boldsymbol{3AS1}$	1.0000	0.6340	1.0000
$\boldsymbol{3AS2}$	0.8181	0.7260	1.0000

The corresponding Gantt charts of these two solutions are shown in Figure 6.1, where the resources assigned to each task are shown next to the corresponding task bars.



(a) Gantt chart of best solution $3AS1$ found by NSGA-II for project $P3A$



(b) Gantt chart of best solution $3AS2$ found by NSGA-II for Project $P3A$

Figure 6.1: Gantt charts of the best solutions found by NSGA-II for project $P3A$ in Approach 3

From Table 6.2, it can be seen that all runs of NSGA-II generate feasible solutions for project $P3A$. Also, the final Pareto fronts represented between one and three unique solutions, something which is actually expected since the very small size of the software project guided individuals to converge to these solutions. The individuals of the final Pareto fronts were then combined to isolate the overall best solutions forming the approximated Pareto front (Figure 6.2).

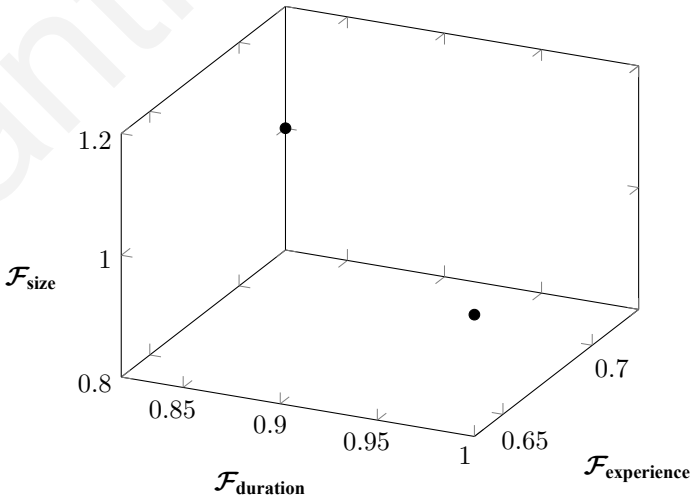


Figure 6.2: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3A$ in Approach 3

Table 6.6 presents the runs in which the two optimal solutions were indeed found together with the corresponding number of individuals representing those solutions in each run's final Pareto front. As can be seen, best solution $3AS1$ was found in the final Pareto front of 12 runs, whereas optimal solution $3AS2$ was found in six. However, only a third of the runs managed to find both solutions (runs 07, 10, 11, 12 and 15). What's more, in the majority of cases where an best solution was found, it was either highly prevalent over the other (as in runs 04, 05, 07, 09, 10, 11, 12 and 15) or the only solution found (as in runs 03, 06, 08 and 13).

Table 6.6: Frequency of the best solutions in the final Pareto fronts generated by NSGA-II for project $P3A$ in Approach 3

	Frequency	
	$\mathbf{3AS1}$	$\mathbf{3AS2}$
Run 03	100	0
Run 04	96	0
Run 05	97	0
Run 06	100	0
Run 07	99	1
Run 08	100	0
Run 09	99	0
Run 10	6	94
Run 11	99	1
Run 12	99	1
Run 13	100	0
Run 14	0	2
Run 15	93	7

It is clear from the results of Table 6.6 that for small-scale software projects the algorithm is able to find the best task schedules and developer allocations. As is reflected in real-world situations, assigning more experienced developers to tasks can create conflicts in the availability of the developers. In order to accommodate this, the algorithm will attempt to delay the execution of the conflicting tasks so that it keeps fixed the assignment of more experienced developers, which consequently increases the overall duration of the project. Such is the case for tasks $T02$ and $T03$ of project $P3A$. In Figure 6.1a, developer $R03$ is assigned to carry out task $T02$ (experience in skill $S02 = 0.36$) and developer $R02$ is assigned to carry out task $T03$ (experience in skill $S03 = 0.74$). This assignment, whilst optimal with respect to duration, it is not optimal with respect to experience since developer $R02$ is more experienced in $S02$ than developer $R03$ ($0.82 > 0.36$). By assigning developer $R02$ to also carry out task $T02$, an assignment conflict is created in the availability of developer $R02$. Therefore, in order to handle this, the algorithm pushes back the execution of task

$T03$, conversely increasing the duration of the software project by three days and generating the new competing best solution shown in Figure 6.1b. It is evident from this example that the two objective functions with respect to duration and experience are competing in nature. Due to the small scale of the software project, the size objective function also tries to minimize the number of developers assigned to each task, therefore selecting only one developer per task.

The AMOSA implementation of the proposed approach produced a total of three unique solutions, which include the two optimal solutions presented in Figure 6.1. Regarding the optimality of the solutions generated in each run, none of the runs managed to successfully generate both optimal solutions, unlike NSGA-II, which managed to do so in five of the runs. In fact, 11 runs generated only one of the best solutions, whereas the remaining four runs generated neither optimal solutions. This demonstrates that even with a small-scale and less complex software project, a genetic algorithm approach is more appropriate than simulated annealing for optimization in the context of this constrained multi-objective problem, since the rate of optimality is more frequent in NSGA-II than in AMOSA. With respect to feasibility, only one run did not manage to provide an archive with any feasible solutions (run 13).

For project $P3B$ various behaviours of the NSGA-II algorithm can be observed by examining the number of unique solutions in the final Pareto front of each run in Table 6.2. First, in some cases the individuals comprising the final Pareto front represent one single unique solution. This occurs in runs 02, 03, 07, 09 and 13. This can be read as possible evidence that the crossover and mutation operators adopted are not capable of untrapping individuals to search for more varied solutions. One possible solution is to gradually increase the rate of exploration of the search space with a higher probability of mutation as the algorithm is executing so that more nondominated solutions could be generated. A second observation is that in certain instances a relatively small number of unique solutions are represented by the individuals of the final Pareto front. Specifically, in runs 01, 04, 08, 12 and 14, between two and seven solutions are represented by individuals in their respective final Pareto front. Furthermore, in some runs population individuals represent many unique solutions, as occurred in runs 05, 06, 10, 11 and 15, which leads to an average number of unique solutions of 21 for project $P3B$. This may be due to more iterations required for the algorithm to converge to a smaller set of best solutions. Finally, all 316 unique solutions that were found by NSGA-II in the 15 runs for project $P3B$ provided feasible task schedules and developer allocations. By carrying out a fast nondominated sort of these solutions, the approximation Pareto

front in the end comprised five best solutions, which listed in Table 6.7 and depicted in Figure 6.3. The frequency of these solutions as individuals in the runs are shown in Table 6.8.

Table 6.7: Objective function values corresponding to the best solutions generated by NSGA-II for project $P3B$ in Approach 3

	Objective function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\mathbf{3BS1}$	0.5815	0.6109	1.0000
$\mathbf{3BS2}$	0.6037	0.6425	1.0000
$\mathbf{3BS3}$	0.6079	0.6717	1.0000
$\mathbf{3BS4}$	0.7350	0.5817	1.0000
$\mathbf{3BS5}$	0.6302	0.6108	1.0000

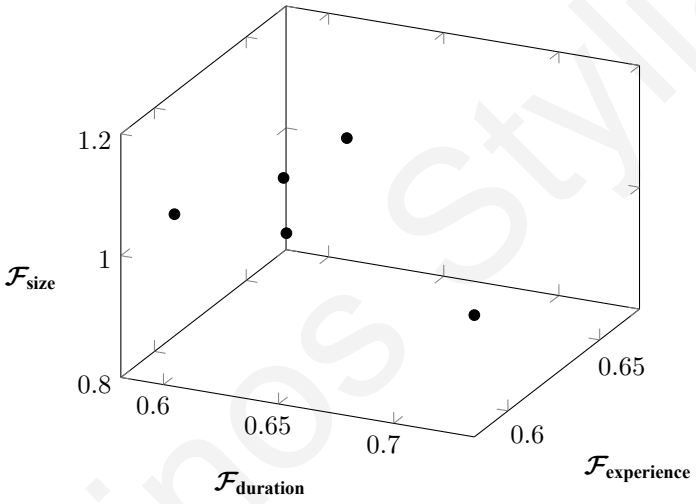


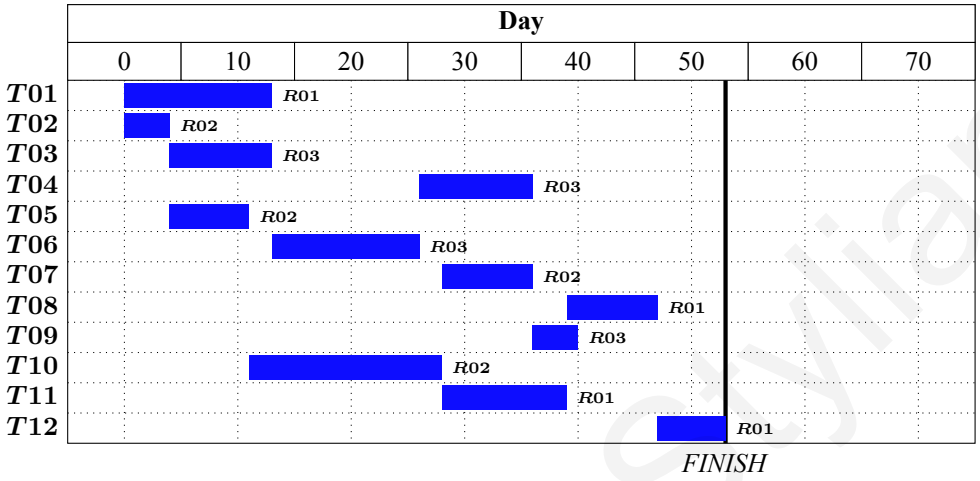
Figure 6.3: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3B$ in Approach 3

Table 6.8: Frequency of the best solutions in the final Pareto fronts generated by NSGA-II for project $P3B$ in Approach 3

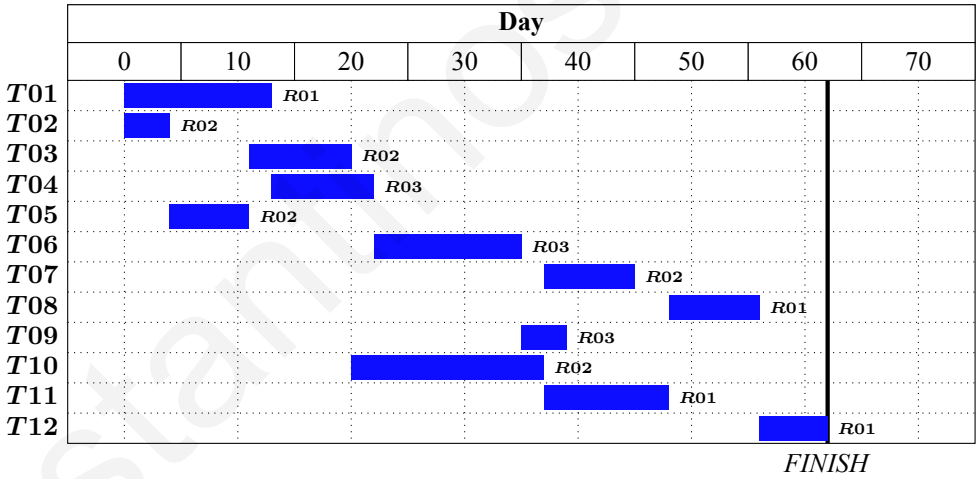
	Frequency				
	$\mathbf{3BS1}$	$\mathbf{3BS2}$	$\mathbf{3BS3}$	$\mathbf{3BS4}$	$\mathbf{3BS5}$
Run 02	100	0	0	0	0
Run 11	0	5	1	0	0
Run 12	0	0	0	89	0
Run 13	0	0	0	0	100

A closer look at the objective function values in Table 6.7 shows that, as anticipated, the shorter the project's make span, the lower the experience. Again, this shows the competitive nature of the objectives as the algorithm tries to balance developer availability conflicts. This is demonstrated by

the schedules of the two (near-)optimal solutions in Figure 6.4. Optimal solution $3BS1$ provides the shortest project duration, while best solution $3BS3$ the most experienced software project team. From a practical perspective, a software project manager looking to maximize resources or minimize make span can use this approach to select the most appropriate project schedule and team assignment from a pool of possible strategies.



(a) Gantt chart of best solution $3BS1$ found for project $P3B$



(b) Gantt chart of best solution $3BS3$ found for project $P3B$

Figure 6.4: Gantt charts of several best solutions found for project $P3B$ in Approach 3

All the solutions produced satisfy all the constraints, thus, generating feasible task schedules and developer allocations. Interestingly, individual solutions in the approximation Pareto front have varying frequencies in their respective final Pareto front. For instance, best solution $3BS3$ is

found only by one individual in run 11, whereas best solution $3BS1$ is represented by all individuals in the final Pareto front of run 02. This may indicate that the random nature of the algorithm particularly during selection, recombination and mutation does not guarantee the convergence to a single (near-)optimal solution or the divergence to a set of (near-)optimal solutions.

Results produced from executing AMOSA on Project $P3B$ showed that, while NSGA-II had no runs that generated only infeasible solutions, the simulated annealing method had two instances where an run's final archive contained individuals violating any or all of the constraints (runs 05 and 14). The average number of unique solutions found per run was 31, which is higher than in the case of NSGA-II (21). Still, the number of best solutions in the approximation Pareto archive was almost the same as for NSGA-II.

In project $P3C$, six NSGA-II runs produced final populations all with feasible solutions, whereas there were nine instances (those marked with an asterisk in Table 6.2) where the runs of the algorithm produced a final population of infeasible solutions that violated at least one of the three constraints. As the problem size and complexity increases the more it is clear that the algorithm starts to find difficulties in generating (near-)optimal solutions whilst trying to cope with avoiding constraint violations. The final populations of the six runs that generated only feasible solutions (specifically, runs 03, 04, 07, 09, 13 and 15) were subsequently combined to yield the eight best solutions presented in Table 6.9 and plotted in Figure 6.5. The frequency of these eight best solutions in their respective runs are given in Table 6.10. While most of the solutions in this front only appeared once in their respective runs, three of the solutions were represented by more than one individual. The corresponding Gantt chart of the most predominant individual, $3CS5$, is shown in Figure 6.6.

Table 6.9: Objective function values corresponding to the best solutions generated by NSGA-II for Project $P3C$ in Approach 3

	Objective function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\mathbf{3CS1}$	0.7536	0.5507	0.7222
$\mathbf{3CS2}$	0.7530	0.5521	0.7000
$\mathbf{3CS3}$	0.6915	0.5521	0.7333
$\mathbf{3CS4}$	0.8194	0.5301	0.7333
$\mathbf{3CS5}$	0.8182	0.5411	0.7333
$\mathbf{3CS6}$	0.4219	0.5532	0.7000
$\mathbf{3CS7}$	0.4208	0.5546	0.7000
$\mathbf{3CS8}$	0.4207	0.5569	0.7333

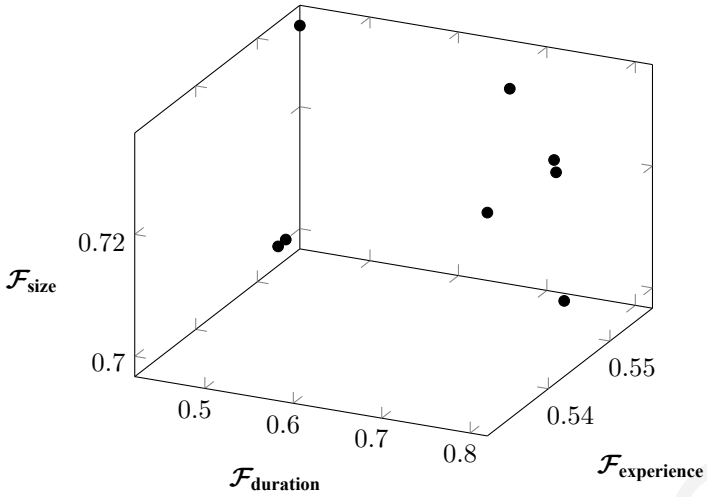


Figure 6.5: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3C$ in Approach 3

Table 6.10: Frequency of the best solutions in the final Pareto fronts generated by NSGA-II for project $P3C$ in Approach 3

	Frequency							
	$\mathbf{3CS1}$	$\mathbf{3CS2}$	$\mathbf{3CS3}$	$\mathbf{3CS4}$	$\mathbf{3CS5}$	$\mathbf{3CS6}$	$\mathbf{3CS7}$	$\mathbf{3CS8}$
Run 04	1	1	1	0	0	0	0	0
Run 07	0	0	0	4	87	0	0	0
Run 15	0	0	0	0	0	6	1	1

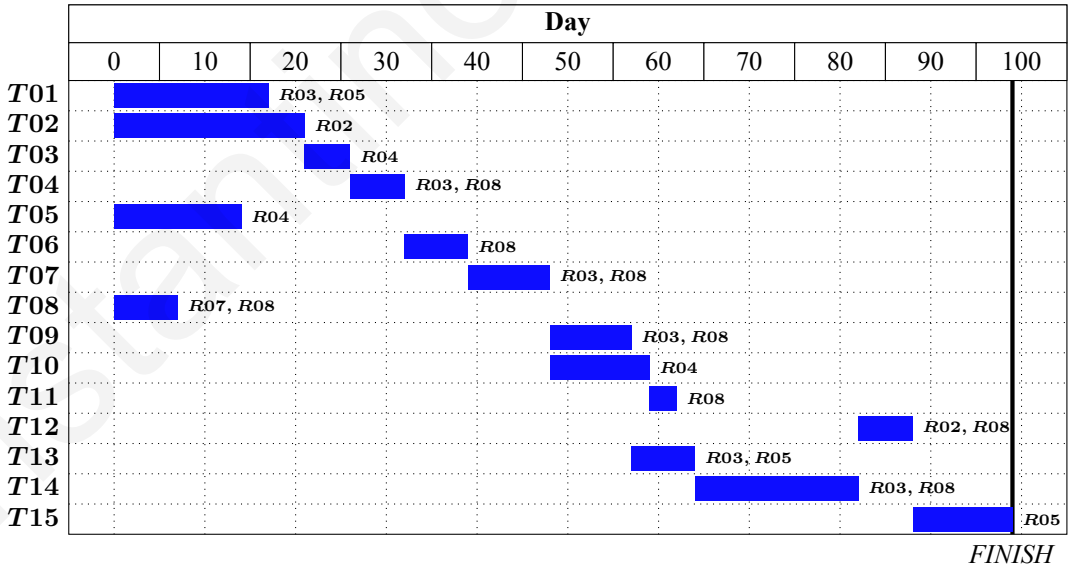


Figure 6.6: Gantt chart of best solution $3CS5$ found for project $P3C$ in Approach 3

Applying the simulated annealing algorithm on project $P3C$ produced some interesting observations. Only two runs produced feasible solutions, which is lower than in the case of NSGA-II, where five runs produced feasible solutions. However, on average, the number of unique solutions at the end of each run is the same (86). This shows that the ability of AMOSA in enforcing diversity in solutions in the context of the described problem is equal with that of NSGA-II. However, after combining the final archives of each run, the number of best solutions in the approximation Pareto archive of AMOSA is only two, compared to the eight in NSGA-II.

Regarding the optimality of the solutions in the final archive, after careful examination of all schedules and allocations generated in both techniques, AMOSA provided far less suitable solutions, having unnecessary gaps and also unnecessary allocation of noncontributing developers to tasks, something which was not observed in the NSGA-II implementation. Furthermore, both the solutions in the approximation Pareto archive were individuals from the same run.

The software project represented by project $P3D$ is comparatively larger than the three previous projects. The results obtained here show that all but one run of NSGA-II managed to contain a final Pareto front of feasible solutions, as opposed to none with AMOSA. Run 09 of the NSGA-II implementation, however, produced a final Pareto front of infeasible solutions and, in particular, all individuals failed to satisfy the dependency violation constraint of Equation (5.5). This may primarily be attributed to the increased number of tasks, as well as number of skills required per task in the software project. Due to the overall higher number of skills needed to be satisfied, in this instance, the algorithm encounters difficulties in satisfying the constraint while simultaneously attempting to optimize the competing functions. By combining the final Pareto fronts of the runs, a total of 215 best solutions were present in the approximation Pareto front, as illustrated in Figure 6.7.

Tables 6.11 and 6.12 show the objective function values and frequency, respectively, of the best solutions of the approximation Pareto front that appeared more than once in their respective final Pareto front. The remaining 209 solutions appearing as individuals just once are omitted.

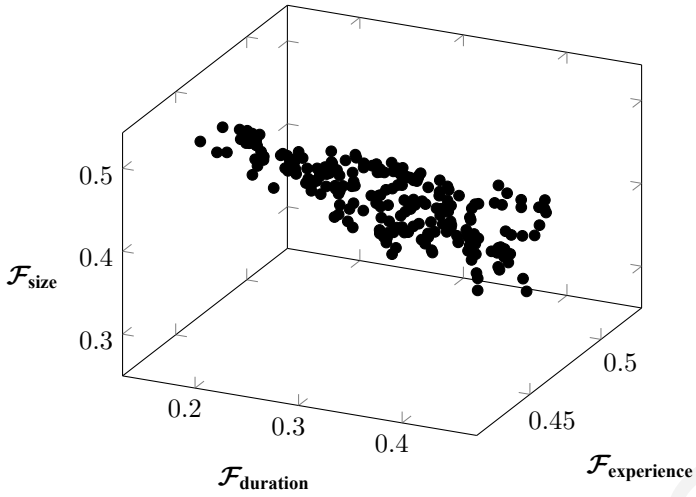


Figure 6.7: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3D$ in Approach 3

Table 6.11: Objective function values corresponding to the best solutions generated by NSGA-II for Project $P3D$ in Approach 3

	Objective function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\mathbf{3DS1}$	0.2299	0.4392	0.5000
$\mathbf{3DS2}$	0.3058	0.5154	0.3366
$\mathbf{3DS3}$	0.3065	0.5150	0.3292
$\mathbf{3DS4}$	0.3059	0.5136	0.3509
$\mathbf{3DS5}$	0.3628	0.4600	0.4213
$\mathbf{3DS6}$	0.2691	0.5262	0.3000

Table 6.12: Frequency of the best solutions in the final Pareto fronts generated by NSGA-II for project $P3D$ in Approach 3

	Frequency					
	$\mathbf{3DS1}$	$\mathbf{3DS2}$	$\mathbf{3DS3}$	$\mathbf{3DS4}$	$\mathbf{3DS5}$	$\mathbf{3DS6}$
Run 02	2	0	0	0	0	0
Run 07	0	2	2	2	0	0
Run 11	0	0	0	0	2	0
Run 14	0	0	0	0	0	2

Upon closer examination of the task schedules of all Pareto front solutions it was established that, similar to the previous approaches, there were unnecessary gaps between dependent tasks. This indicates that the algorithm was not able to find (near-)optimal solutions in any of the runs. An example of such nonoptimal solution appearing in the approximated Pareto front can be seen in the Gantt chart of Figure 6.8. From the figure, task $T18$ starts its execution on Day 124 while its

predecessor (task $T16$) finishes its execution on Day 105. This means that there are 20 days in the project where developers are idle. What is expected of the algorithm is to shift the start day of task $T18$ towards the finishing day of task 16, thus minimizing the duration of the project. This also occurs between tasks $T08$ and $T09$. Although there is no dependency between the two tasks here, there is, nevertheless, a gap of 4 days where developers $R01$ and $R06$ are expecting to commence work on task $T09$ after developer $R01$ is freed by the completion of task $T08$. Either or both of these types of situations occurred in all of the solutions of the approximation Pareto front, which gives an indication of the difficulties challenging the algorithm to generate (near-)optimal solutions when the size of the software project increases in addition to trying to handle constraints.

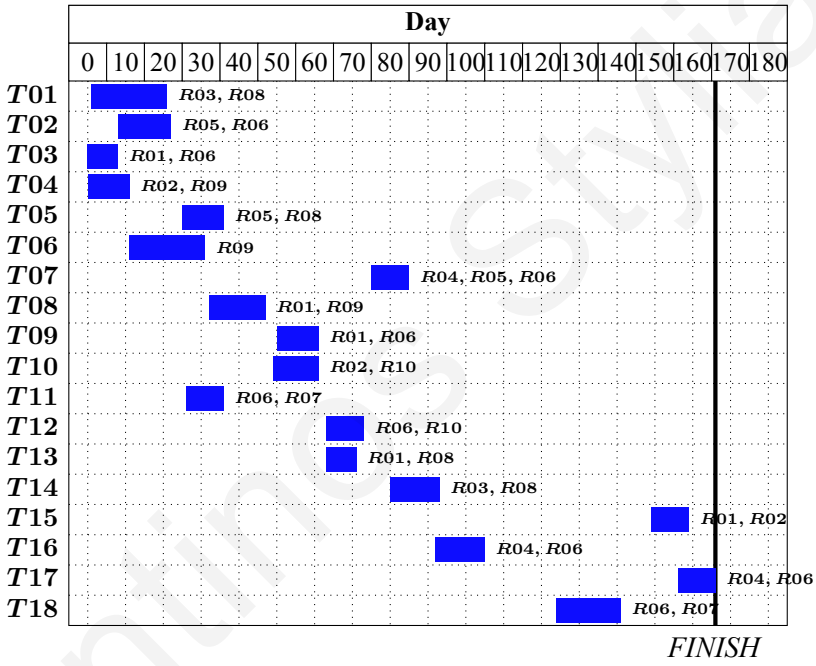


Figure 6.8: Gantt chart of best solution $3DS5$ found for project $P3D$ in Approach 3

In comparison with simulated annealing, only a single solution made up the approximation Pareto archive, and this solution was infeasible. The level of diversity of the solutions in each run is, however, almost just as high as with NSGA-II. The fact that no feasible solutions were generated in any of the runs is opposite to the behaviour of the NSGA-II algorithm, which managed to produce feasible solutions in all but one run. This is possibly attributed to the fact that, as a local search technique, the method encounters difficulty in evolving in a direction of a more feasible solution in a search space with multiple dimensions and constraints.

In the case of project $P3E$, only run 07 of the NSGA-II algorithm managed to produce a final Pareto front with feasible solutions. The runs which produced infeasible solutions all failed to satisfy the dependency satisfaction constraint (Equation (5.5)), while several also failed to satisfy the assignment validity constraint (Equation (5.9)). Clearly, the larger number of tasks coupled with the increased number of dependencies and small number of developers available to staff each task make the algorithm unable to sufficiently cope with balancing the objectives. The individuals of the final Pareto front of run 07 make up the approximation Pareto front for NSGA-II (Figure 6.9). The objective function values of these six best solutions are shown in Table 6.13.

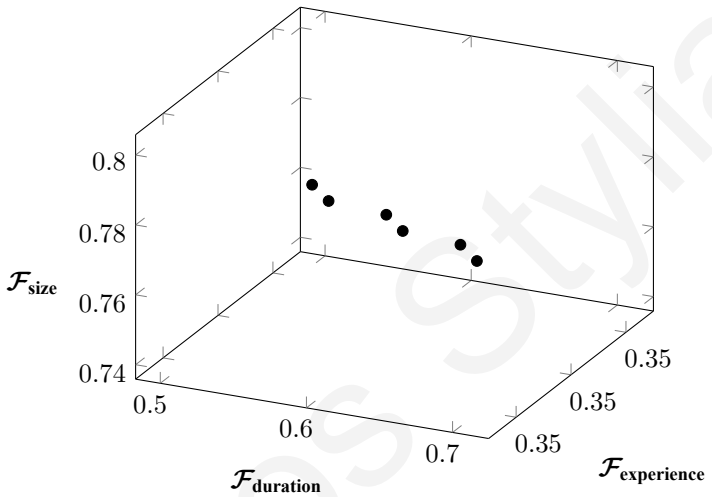


Figure 6.9: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3E$ in Approach 3

Table 6.13: Objective function values corresponding to the best solutions generated by NSGA-II for Project $P3E$ in Approach 3

	Objective function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\boldsymbol{3ES1}$	0.6039	0.3470	0.8000
$\boldsymbol{3ES2}$	0.6039	0.3476	0.7917
$\boldsymbol{3ES3}$	0.6039	0.3497	0.7750
$\boldsymbol{3ES4}$	0.6039	0.3524	0.7500
$\boldsymbol{3ES5}$	0.6039	0.3503	0.7667
$\boldsymbol{3ES6}$	0.6039	0.3530	0.7417

The important observation made here is that the scheduling of the project tasks is the same for all solutions and, hence, all schedules have a duration of 134 days. Even so, the algorithm tries to balance the competition between the experience objective function and the size objective function.

An increase in the value of one objective decreases the value of the other objective. Decreasing the size of a team in a task means that more developers become available to participate in other tasks and possibly increasing the experience value of those other tasks. The approximation Pareto front also shows the trade-off between these two objectives. What's more, the solutions generated are more-or-less equally represented by individuals in run 07, at roughly 16–17 appearances each (Table 6.14). The Gantt chart in Figure 6.10 shows the resulting task scheduling represented by best solution $3ES6$. Again, the presence of unnecessary gaps between tasks can be observed, causing extra time to be added to the project's duration. A possible solution to eliminate gaps could come in the form of a technique that allows the algorithm to explore the surrounding problem space more effectively in order to escape from local optima, and by allowing more evolutions of the algorithm.

Table 6.14: Frequency of the best solutions in the final Pareto fronts generated by NSGA-II for project $P3E$ in Approach 3

	Frequency					
	$\mathbf{3ES1}$	$\mathbf{3ES2}$	$\mathbf{3ES3}$	$\mathbf{3ES4}$	$\mathbf{3ES5}$	$\mathbf{3ES6}$
Run 07	17	17	16	16	17	17

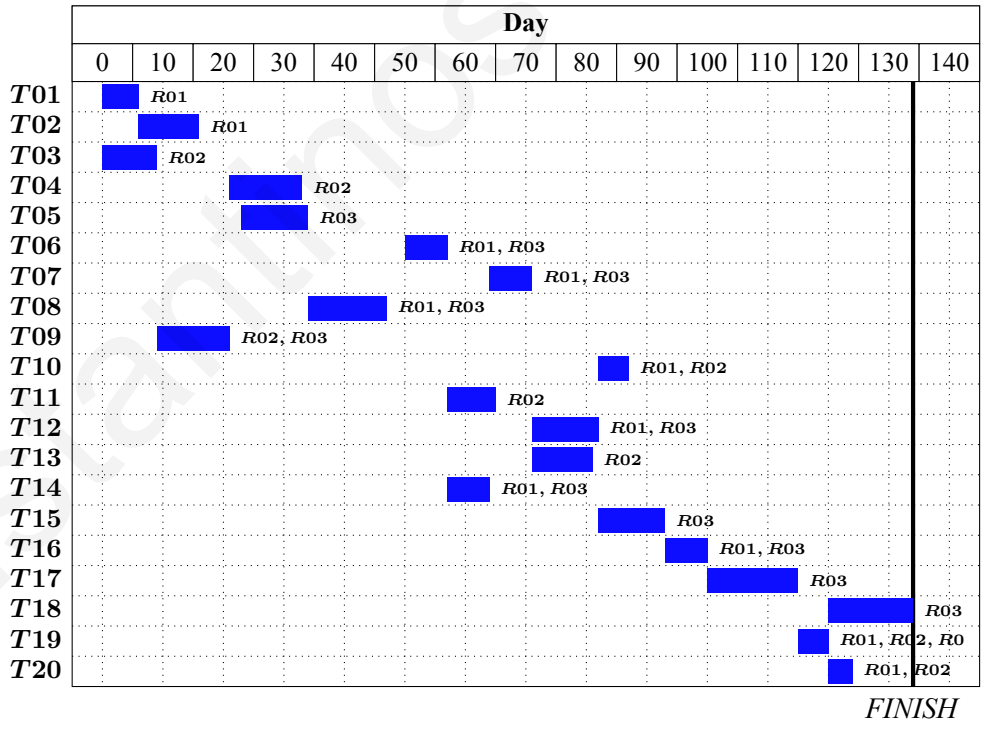


Figure 6.10: Gantt chart of best solution $3ES5$ found for project $P3E$ in Approach 3

The simulated annealing approach behaved similarly in project $P3E$ as in project $P3D$, generating a single infeasible solution in the approximation Pareto archive, with no runs generating any feasible solutions. However, the average number of unique solutions per run in this case was slightly lower in AMOSA than in NSGA-II.

The runs of project $P3F$ by NSGA-II were marginally more successful in producing final individuals with feasible solutions than project $P3E$. By comparing the project dimensions of the two instances in Table 6.1, some indications can be given as to why this occurs. First, in project $P3F$ there are quite more developers available to be assigned to tasks and also they possess experience in a higher number of skills on average than in project $P3E$. For this reason, the algorithm should be able to handle the skills fulfilment constraint Equation (5.7) and the developer assignment validity constraint Equation (5.9) more easily since there are more developers to choose from with a wider variety of skill sets. Second, project $P3F$ has a lower dependency rate meaning that, on average, fewer dependency relationships exist between the total number of possible dependencies among the tasks, which are one-and-a-half times as much as those of project $P3E$. Consequently, the algorithm should be able to satisfy the dependency satisfaction constraint Equation (5.5) with less difficulty in project $P3F$ than in project $P3E$. On this basis, it is expected that final Pareto fronts with feasible solutions should be more frequent in runs of project $P3F$ than in project $P3E$, yet the majority produced infeasible solutions. One possible reason for this may be related to the dependency relationships between tasks. In project $P3F$, the dependency relationship rate of the software project is relatively lower than in the project $P3E$, but nevertheless the number of dependency relationships is higher. This suggests that using the rate alone as an indicator of complexity may not be sufficient. A closer examination of the dependency relationships in project $P3F$ revealed that there are actually several tasks that have a high number of predecessor tasks in contrast with project $P3E$. Due to tasks being heavily dependent on others, as the algorithm tries to balance the objectives, it would more likely encounter problems in satisfying the dependency violation constraint, as well as the developer assignment conflict constraint. This, therefore, can provide an explanation as to why the algorithm is not able to produce more feasible solutions in this instance.

The feasibility of the solutions using the simulated annealing method also suffered from this, as it again failed to provide any solutions that did not violate all of the constraints imposed. It should be noted, though, that the infeasible solution of the approximation Pareto archive violated

the dependency constraint (at a value of 0.0308) and the developer availability constraint (at a value of 0.0179). Considering that a feasible solution will have constraint values of zero, the level of infeasibility of the solution is relatively small, which indicates that the simulated annealing algorithm in this run was close to producing at least one feasible solution.

In NSGA-II, runs 04 and 05 for project $P3F$ were the only ones that managed to provide final Pareto fronts with feasible solutions. After joining the two populations, the approximation Pareto front consisted of 50 individuals (which, incidentally, all belonged to the final Pareto front of run 05) as shown in Figure 6.11.

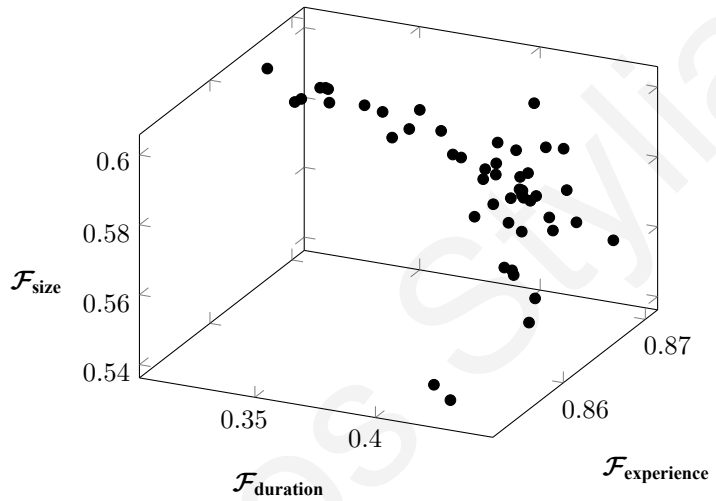


Figure 6.11: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3F$ in Approach 3

The fact that a small number of runs with feasible populations were generated can be attributed, perhaps, to the fact that project $P3F$ has a considerably higher number of tasks, which on average require more skills to be carried out. While this does not directly affect the satisfaction of constraints in theory, in practice the algorithm has to conduct exploration and exploitation of a much larger search space in order to find (near-)optimal solutions and, by doing so, possibly compromises the feasibility of its solutions. Out of the 50 solutions comprising the approximation Pareto front, 11 were represented by more than one individual in their respective runs final population. For the sake of brevity, only the objective function values and frequencies of these individuals are displayed in Tables 6.15 and 6.16, respectively.

Table 6.15: Objective function values corresponding to the best solutions generated by NSGA-II for Project $P3F$ in Approach 3

	Objective function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\boldsymbol{3FS1}$	0.4144	0.8543	0.5420
$\boldsymbol{3FS2}$	0.4111	0.8648	0.5553
$\boldsymbol{3FS3}$	0.4311	0.8514	0.5448
$\boldsymbol{3FS4}$	0.4228	0.8659	0.5698
$\boldsymbol{3FS5}$	0.4002	0.8685	0.5817
$\boldsymbol{3FS6}$	0.3506	0.8668	0.5900
$\boldsymbol{3FS7}$	0.3586	0.8677	0.5844
$\boldsymbol{3FS8}$	0.3906	0.8689	0.5761
$\boldsymbol{3FS9}$	0.3241	0.8681	0.5872
$\boldsymbol{3FS10}$	0.3232	0.8679	0.5917
$\boldsymbol{3FS11}$	0.3060	0.8692	0.5833

Table 6.16: Frequency of the best solutions in the final Pareto fronts generated by NSGA-II for project $P3F$ in Approach 3

	Frequency					
	$\boldsymbol{3FS1}$	$\boldsymbol{3FS2}$	$\boldsymbol{3FS3}$	$\boldsymbol{3FS4}$	$\boldsymbol{3FS5}$	$\boldsymbol{3FS6}$
Run 05	3	2	2	2	3	2
		$\boldsymbol{3FS7}$	$\boldsymbol{3FS8}$	$\boldsymbol{3FS9}$	$\boldsymbol{3FS10}$	$\boldsymbol{3FS11}$
Run 05		3	2	2	3	2

Another interesting observation here is that the competing nature of the objective functions is not clear, that is, an increase in one value does not imply a decrease in the other(s). This, once more, is possibly due to the fact that the software project consists of a large number of tasks and, thus, the algorithm is not able to provide (near-)optimal solutions but instead yields schedules with unnecessary delays between tasks. Had the algorithm managed to overcome this, then the values of the corresponding objective function would have been lower (leading to a lower make span) and more reflective of the competitiveness of the objective functions.

For project $P3G$, three of the NSGA-II runs managed to produce final Pareto fronts with feasible solutions. Compared to project $P3F$, the software project here had a higher number of dependency relationships for the algorithm to deal with, in addition to much lower number of available developers. This combination could be a possible factor determining whether the algorithm is able to handle such high complexity in its attempt to optimize the objectives, especially with limited resources. Furthermore, by studying the runs with infeasible solutions, it was observed again that the individuals in each of these populations only breached the dependency violation constraint. The

143 solutions comprising the approximation Pareto front of NSGA-II for project $P3G$ can be seen in Figure 6.12, while the objective function values of the most frequent are shown in Table 6.17.

Table 6.17: Objective function values corresponding to the best solutions generated by NSGA-II for Project $P3G$ in Approach 3

	Objective function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\mathbf{3GS1}$	0.5162	0.5775	0.6250
$\mathbf{3GS2}$	0.4677	0.5719	0.6625
$\mathbf{3GS3}$	0.4618	0.5733	0.6667
$\mathbf{3GS4}$	0.2904	0.5932	0.6292
$\mathbf{3GS5}$	0.3125	0.5903	0.6542
$\mathbf{3GS6}$	0.3425	0.5742	0.6917
$\mathbf{3GS7}$	0.4080	0.5851	0.5708
$\mathbf{3GS8}$	0.3565	0.5962	0.6000
$\mathbf{3GS9}$	0.3049	0.5980	0.5875
$\mathbf{3GS10}$	0.3243	0.5831	0.6767
$\mathbf{3GS11}$	0.3365	0.5783	0.6833
$\mathbf{3GS12}$	0.3608	0.5701	0.6708
$\mathbf{3GS13}$	0.3420	0.5511	0.6917
$\mathbf{3GS14}$	0.4555	0.5555	0.6583
$\mathbf{3GS15}$	0.4501	0.5453	0.6778
$\mathbf{3GS16}$	0.4204	0.5580	0.6917
$\mathbf{3GS17}$	0.4662	0.5699	0.5788
$\mathbf{3GS18}$	0.4296	0.5775	0.5719
$\mathbf{3GS19}$	0.4045	0.5820	0.5792
$\mathbf{3GS20}$	0.4129	0.5805	0.5875
$\mathbf{3GS21}$	0.4351	0.5795	0.6042

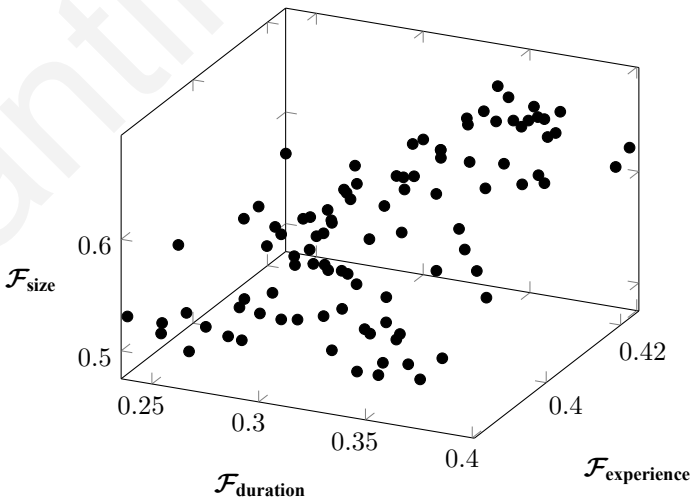


Figure 6.12: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3G$ in Approach 3

The corresponding objective function values are shown in Table 6.18. Because the algorithm was unable to provide (near-)optimal solutions in any of the runs, the competing nature of the objective functions is again not clear in this project. AMOSA, on the other hand, again failed to produce any feasible solutions in the 15 runs.

Table 6.18: Frequency of the best solutions in the final Pareto fronts generated by NSGA-II for project $P3G$ in Approach 3

	Frequency						
	$\mathbf{3GS1}$	$\mathbf{3GS2}$	$\mathbf{3GS3}$	$\mathbf{3GS4}$	$\mathbf{3GS5}$	$\mathbf{3GS6}$	$\mathbf{3GS7}$
Run 07	2	2	2	0	0	0	0
Run 10	0	0	0	2	2	2	2
Run 11	0	0	0	0	0	0	0
	$\mathbf{3GS8}$	$\mathbf{3GS9}$	$\mathbf{3GS10}$	$\mathbf{3GS11}$	$\mathbf{3GS12}$	$\mathbf{3GS13}$	$\mathbf{3GS14}$
Run 07	0	0	0	0	0	0	0
Run 10	2	2	2	2	0	0	0
Run 11	0	0	0	0	2	2	2
	$\mathbf{3GS15}$	$\mathbf{3GS16}$	$\mathbf{3GS17}$	$\mathbf{3GS18}$	$\mathbf{3GS19}$	$\mathbf{3GS20}$	$\mathbf{3GS21}$
Run 07	0	0	0	0	0	0	0
Run 10	0	0	0	0	0	0	0
Run 11	2	2	2	2	2	2	2

The eighth and final project, project $P3H$ was similar to project $P3G$, but with a much lower level of complexity among dependencies. Immediately noticeable is the fact that all but one run of NSGA-II terminated with a final Pareto front of only feasible solutions, which is contrary to projects $P3E$, $P3F$ and $P3G$, where very few of the runs were able to find feasible solutions successfully. The reason for this may be due to not having too many dependency relationships between tasks, the dependency violation constraint is easier to handle by the algorithm and, consequently, the objective functions can be optimized more straightforwardly as shown by the nine solutions of the final approximation Pareto front in Figure 6.13 and Table 6.19.

The frequency of each solution is displayed in Table 6.20, which shows that solutions in the approximation Pareto front are only represented once by the individuals in their respective run's final Pareto front.

The results obtained from employing simulated annealing as the optimization approach are similar to those of projects $P3F$ and $P3G$, where only one infeasible solution comprised the final combined archive, demonstrating that NSGA-II outperforms AMOSA when applied to carry out optimization for the proposed approach.

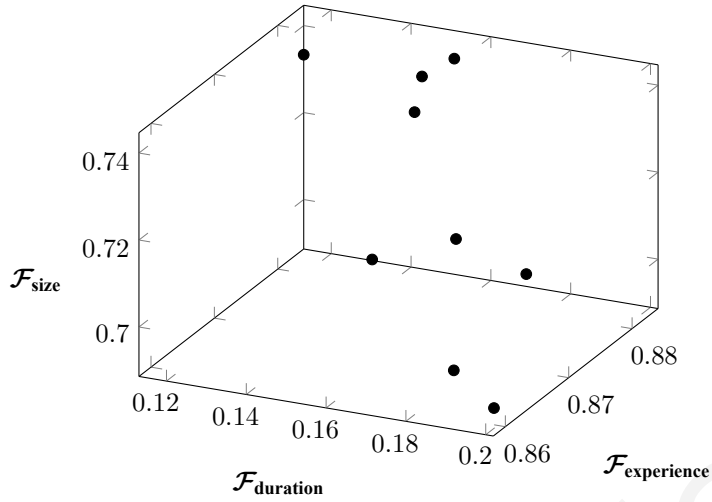


Figure 6.13: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P3H$ in Approach 3

Table 6.19: Objective function values corresponding to the best solutions generated by NSGA-II for Project $P3H$ in Approach 3

	Objective Function		
Solution	$\mathcal{F}_{\text{duration}}$	$\mathcal{F}_{\text{experience}}$	$\mathcal{F}_{\text{size}}$
$\boldsymbol{3HS1}$	0.1781	0.8672	0.7200
$\boldsymbol{3HS2}$	0.1131	0.8841	0.7333
$\boldsymbol{3HS3}$	0.1529	0.8828	0.7400
$\boldsymbol{3HS4}$	0.1512	0.8776	0.7333
$\boldsymbol{3HS5}$	0.1509	0.8711	0.7067
$\boldsymbol{3HS6}$	0.1510	0.8789	0.7400
$\boldsymbol{3HS7}$	0.1920	0.8581	0.7022
$\boldsymbol{3HS8}$	0.2000	0.8594	0.6933
$\boldsymbol{3HS9}$	0.2019	0.8633	0.7200

Table 6.20: Frequency of the best solutions in final Pareto fronts generated by NSGA-II for project $P3H$ in Approach 3

	Frequency								
	$\boldsymbol{3HS1}$	$\boldsymbol{3HS2}$	$\boldsymbol{3HS3}$	$\boldsymbol{3HS4}$	$\boldsymbol{3HS5}$	$\boldsymbol{3HS6}$	$\boldsymbol{3HS7}$	$\boldsymbol{3HS8}$	$\boldsymbol{3HS9}$
Run 01	1	0	0	0	0	0	0	0	0
Run 04	0	1	0	0	0	0	0	0	0
Run 05	0	0	1	0	0	0	0	0	0
Run 14	0	0	0	1	1	1	0	0	0
Run 15	0	0	0	0	0	0	1	1	1

6.6 Summary

The most important aspect of this attempt is the introduction of a Pareto ranking mechanism in order to suggest a set of solutions from which a project manager can select to adopt based on his or her preferences and criteria, rather than having a weighted-sum approach requiring a project manager to decide beforehand the importance of each objective to provide only one solution.

Through investigation of the results obtained from the runs, it was observed that the NSGA-II algorithm performed better in cases consisting of small- and medium-sized software projects. In these cases, the algorithm was able to produce feasible solutions that satisfied all constraints, in addition to finding diverse (near-)optimal task schedules and developer allocations. However, with an increase in the number of tasks and number of dependency relationships the algorithm found difficulties in producing feasible solutions, especially in cases where the number of available developers and the skills possessed by the developers was low.

The main obstacle lies in the algorithm's inability to avoid task dependency violations and developer assignment conflicts whilst trying to handle the competitive nature of the objective functions. Furthermore, in runs where feasible solutions were indeed produced, the algorithm only managed to generate (near-)optimal schedules and/or allocations. These (near-)optimal solutions continued to suffer from unnecessary gaps in their represented schedules, which was a problem also identified in the previous two approaches. One way to handle this could be to introduce a forced-correction step in the algorithm after a certain number of iterations. This step would be responsible for selecting population individuals with unnecessary gaps and removing them from within their corresponding schedule. Then these newly-corrected individuals would be reinserted back to the population to continue the evolution procedure. Alternatively, treating such individuals as infeasible could be a possible extension to the project duration objective function, whose purpose is to minimize these idle gaps between tasks. However, a more radical yet effective approach could be to change the representation of solutions so that delays between tasks would not be allowed to exist. Specifically, as described in the following fourth approach, the task schedule information could be encoded using a permutation (order) of task run rather than an integer encoding denoting task start days. In this way, the duration of a project can be calculated directly by identifying the finish day of the task that ends the latest.

Apart from fixing the issue of delays between tasks, it is also important to consider how developer allocations affect the overall schedule of a project. The approaches up until now differentiate

developers based on their experience in skills in an attempt to stress the fact that realistically developers are not interchangeable. However, what the approaches do not consider is the effect that this experience has on the time it takes to complete a task – regardless of the developer(s) assigned to a task, the duration is always the same. This is misleading because in reality a project manager does not estimate the duration of a task (in calendar terms) but rather the effort required for the task to be complete. Hence, a highly-skilled developer may complete a task quicker than a less-skilled developer.

Consequently, the fourth approach, presented in Chapter 7 redesigns the two objective functions of this approach by incorporating developer experience as the part of the project duration objective function in the form of productivity. It also introduces the cost of a software project as an objective, which is a more practical criterion for software project managers.

Chapter 7

Approach 4: Optimizing Project Duration and Project Cost using a Pareto Ranking Genetic Algorithm

This approach [93] introduced a number of significant changes to the approaches described previously in Chapters 4–6. First, it aims to minimize the cost of a software project with a new objective function that considers the salary of developers. Second, it adopts an improved representation of the problem in order to handle the issue of unnecessary gaps existing between tasks and, subsequently, to improve the quality of the solutions generated with regards optimality. Third, it replaces the experience of developers in skills with the rate of productivity of developers in professions. It uses this information in conjunction with the effort required by tasks to compute the duration of each task. Hence, the definition of the project duration objective function and the calculation of its corresponding value are revised accordingly to reflect this improvement. As stated in the previously described attempts, the approach is intended to be used for allocating resources and scheduling tasks at the beginning of a software development project to help managers make more accurate budget and time estimates by selecting from a set of possible viable solutions. For this reason, performing multiobjective optimization using Pareto ranking rather than a weighted-sum method was kept.

This attempt takes into account the fact that even though the amount of work required to complete a task does not depend on the developers assigned, the amount of time required to complete a task does. In other words, the effort required for a task will be the same regardless of who is assigned to carry out the task, whereas the duration will vary based on the developers assigned to work on it. Existing optimization attempts have overlooked this, and instead either regard developers as interchangeable (meaning that they all possess the same skills) or focus simply on whether or not

developers possess the necessary skills required by a task, without considering the effect on duration. For example, Antoniol et al. [73], Ren et al. [76], and Di Penta et al. [75] attempt to schedule work packages and allocate teams to work packages with the goal of minimizing project duration. These attempts focus on software maintenance projects, where work packages are assigned teams of developers as they occur in time or are postponed until developers with the required expertise are available again. The latter attempt also focuses on minimizing the idle times of developers, that is, the time a developer waits to be reassigned to another task. However, these attempts consider that teams are equally capable of carrying out tasks and require the same amount of time to do so. Alba and Chicano [64, 65] propose an approach to allocate developers and schedule tasks in order to minimize the cost and duration of software projects. The authors employ a genetic algorithm to perform the optimization, in which the duration of tasks is determined by the degree of dedication of each assigned developer as long as the skill requirements of tasks are satisfied. Using the same approach, Xiao et al. [79] made a comparison between genetic algorithms and particle swarm optimization, and their results show that the latter technique yield better solutions. Also, Minku et al. [94, 95] attempt to improve the quality of solutions and hit rates of the original approach by normalizing the degree of dedication of developers and incorporating a new penalty for evaluating cost and completion time. The approach, however, considers that all developers with a particular skill will possess it to the same level and, thus, assumes that those developers are interchangeable.

In reality, however, developers possess varying levels of skills depending on the knowledge acquired through education and training, their natural abilities and talents, as well as their experiences accumulated over time. Hence, differences in levels of skill implies differences in productivity rates of developers, which in turn affects the duration and cost of each task, and of the project as a whole. Therefore, to take this into account, the new approach moves away from optimizing developers' experience in required skills and, instead, substitutes the effect of experience with the rate of productivity in the respective profession that the skill falls under, similar to Kapur et al. [70] and Ngo-The and Ruhe [71]. The authors here use integer linear programming together with genetic algorithms in order to assign resources and schedule the implementation of features in incremental software development. One of the goals of the optimization is to maximize productivity on the assumption that developers with different levels of skills will naturally have different rates of productivity. This can be used by software development companies as a way to schedule product releases with selected features that lead to an optimum business value.

However, one of the assumptions is that only one developer can be assigned to implement a feature, which may not be practical considering that larger software projects may require two or more developers to work together on a task in any phase of development. In contrast, the approach described in this chapter, as in the three previously described approaches, intentionally allows for more than one developer to be allocated to a task.

In the resource allocation and task scheduling approach proposed by Yanibelli and Amandi [77], the authors take into account the level of effectivity of software developers by assessing the degree to which developers will be effective when assigned to work together on the same task. This information is then used in a genetic algorithm approach that attempts to maximize the effectivity of assigned resources. This approach was later modified, first, with a memetic algorithm [96] and, second, with a diversity-adapted simulated annealing method [97] as a way to improve to the quality of the generated solutions. Make span minimization was also introduced by the authors as an additional criterion to the original approach, hence, transforming it into a multiobjective optimization approach [98]. This was then expanded with the integration of simulated annealing in order to improve the exploitation and exploration search processes of the genetic algorithm [99]. All of these approaches assume that the number of developers required for each task and the level of effectivity between combinations of developers is known in advance, which may pose a problem when two or more developers are assigned together for the first time. Furthermore, the authors completely ignore the cost dimension when allocating and scheduling developers. The most significant downside to this approach, however, is that the duration of each task is not actually influenced by how developers work together. In fact, regardless of how effective the developers assigned to a task are, the duration remains unaffected. The new approach presented here helps managers consider how the contributions of each individual team member are combined with respect to their productivity rate, rather than their effectivity. Specifically, the approach adopts Steiner's classification of task interdependence [100] to help select which operator to use for aggregating individual productivity rates depending on the specific type of software development task.

Finally, another important factor that also influences the duration and cost of a project, which is often neglected, is the issue of communication overhead. Very few attempts have taken into account the increase in time that could occur when developers work together on a task. One attempt is presented by Di Penta et al., who explore how different models and levels of communication overhead affect the allocation of developers in teams, as well as on the overall make span of a

project in a search-based project staffing and scheduling approach [101]. This approach adopts a similar rationale by incorporating communication overhead in the calculation of a task's duration that is then used in the optimization process.

7.1 Research questions

Considering the modifications made to the optimization criteria and also to the definition and representation of the problem, this approach attempted to include harder and more realistic assumptions and constraints concerning the availability and suitability of developers. These assumptions add significantly to the complexity of the problem, which makes it that much harder for the optimization process to find feasible and (near-)optimal solutions. Therefore, the first research target involved adopting several different variations of multiobjective genetic algorithms to carry out the optimization in order to assess their performance. **RQ4.1: How do different multiobjective genetic algorithm variations perform in terms of generating (near-)optimal solutions with respect to this approach for resource allocation and task scheduling?**

Given the fact that software systems progressively become larger as current technology capabilities are improved, it is also important for the approach to be applicable to varying sizes of projects undertaken by varying sizes of software development companies. Consequently, the second research target involved examining the issue of scalability. **RQ4.2: How do different multiobjective genetic algorithm variations behave in terms of scalability as the number of tasks and developers increases in this approach for resource allocation and task scheduling?**

Furthermore, it is equally important to investigate the approach in real-world settings. In particular, it is necessary to study the practical effects of productivity and interdependence type in order to better understand the implication of these attributes in real-world software development, and also to test the general behaviour of the approach when applied to real-world software development projects. **RQ4.3: What observations can be made from the application of the approach in real-world software projects?**

To answer these research questions, a number of generated software project instances were used, as well as several real-world projects.

7.2 Problem description

The RCPSP described in Section 4.2 is revised in order to reflect the new factors taken into consideration in this new attempt.

A software development project consists of a set $T = \{t_1, t_2, \ldots, t_m\}$ of m tasks, which are determined by the project manager based on the activities from the different phases of software development that will be required. All tasks must be undertaken by at least one software developer employed by the development company. The project manager also determines the set $H = \{h_1, h_2, \ldots, h_q\}$ of q professions whose associated skills will be required by the software project. Each task $t_i \in T$ is subsequently assigned one (and only one) of these professions, denoted by t_i^{prof}, the skills of which are required for the task to be completed. For example, a project manager would assign a task that entails interviewing stakeholders to a requirements analysis profession since he or she will determine that elicitation skills will be necessary to carry out the task.

Next, the software project manager determines which tasks are related to each other in the form of dependency relationships. The approach assumes that only finish-to-start dependency relationships exist, meaning that in order for a task to start, all its predecessor tasks must first finish. The set of dependency relationships D contains pairs of tasks such that $(t_i, t_j) \in D$ if task t_j depends on task t_i.

Once the tasks and dependency relationships have been identified, it is up to the project manager to provide an estimate of the effort or workload that will be required to carry out each task. The effort required for each task t_i is denoted by t_i^{effort}. According to the Project Management Institute [102], effort is defined as 'the number of units of labour needed to complete a scheduled activity or work breakdown structure component'. It is commonly expressed in person-hours (that is, the number of hours needed for an average developer to carry out the work), though it is possible to represent effort in person-days, person-weeks or even person-months for large projects.

The human resources of a development company form the set $R = \{r_1, r_2, \ldots, r_n\}$ of n software developers, who are able to participate in a project based on their availability and area of expertise. Each developer $r_j \in R$ is associated with an hourly wage rate (r_j^{salary}), used to calculate the cost of each task to which developer r_j is assigned. The hourly wage rate can be easily obtained from the company's human resource or accounting department.

Often, developers may possess appropriate skills in more than one profession. As a result, they can work on a project in different capacities, thus adding to the complexity of the allocation

and scheduling process. This is particularly true in small-to-medium-sized companies, where developers are often forced to undertake several roles in the company due to limited resources. To accommodate this in this approach, each developer r_j is assigned a rate of productivity for each profession $h_l \in H$ required by the software project. This information is represented by a productivity matrix $PROD = [prod_{jl}]$ where $prod_{jl}$ denotes the rate at which developer r_j is able to carry out any task belonging to profession h_l. The value of this productivity rate is selected by the project manager from the range $[0.0, 2.0]$ and can be determined in any number of ways using the metrics employed by the software development company and then normalized to fit this range. If a development company is mature enough, project managers may be able to adopt principles from standards and frameworks, such as the People CMM [11], which provide ways to quantify productivity using factors such as experience, competency and capacity. Also, project managers may have access to developers' scores in evaluation reports to help determine their productivity rate. Additionally, project managers can use their experience and expertise to assess the technical skills, know-how and performance of developers in past projects with similar tasks to form a productivity profile of their resources. If a developer does not possess any of the skills of a particular profession, then the project manager will assign a rate of productivity equal to 0.0, and the developer will not be considered as a candidate to be assigned to any task belonging to that profession. For an average developer that possesses the skills of a specific profession, the project manager will likely assign a nominal value of 1.0 as a productivity rate for that profession. In the case of a novice or newly-recruited developer that is considered to be half as productive as an average developer in a certain profession, he or she may be assigned a productivity rate of around 0.5 for that profession. On the other hand, for an above-average developer that possesses the skills of a certain profession (such as an expert), a project manager may assign the developer a productivity rate of 2.0 if he or she is considered to be twice as productive as an average developer in that profession.

As mentioned previously, this attempt allows for more than one developer to be assigned to a task. Therefore, it is important to identify how developers will work together to produce the output of each task. In order to do this, the task interdependence categorization proposed in Steiner's taxonomy of group task [100] is adopted. Steiner identified different "combinatorial strategies" that define the ways with which a team's overall contribution to a task can be measured based on the individual contributions of its members. According to Steiner, there are five different types of task interdependence: additive, compensatory, disjunctive, conjunctive and discretionary, where

each type adopts a unique function for aggregating individual productivity in order to determine overall team productivity. The types of task interdependence and their corresponding aggregation functions used in the approach are additive, disjunctive and conjunctive tasks. Hence, the software project manager assigns a task interdependence type (t_i^{type}) to each task $t_i \in T$ taken from set $Y = \{additive, disjunctive, conjuctive\}$.

In the case of additive tasks, the overall productivity of a team is obtained by adding together the individual productivity of its members. Additive tasks are classed as divisible and maximizing, meaning that these tasks can be broken into subtasks, the goals of which focus on the quantity of the output. Examples of additive software development tasks include verification and validation tasks, such as usability inspections and software reviews, where the work can be divided into subtasks in order for developers to attempt to find as many defects as possible [103]. Since developers will work individually in such tasks and then pool together their work, the higher their individual productivity rates, the higher the effectiveness of the team in defect detection. Software development tasks that require brainstorming may also be considered additive tasks. With regards to disjunctive tasks, the overall productivity of a team is equivalent to the highest individual productivity. Disjunctive tasks are unitary and optimizing, indicating that they cannot be further decomposed and that they focus on the output's quality. Database design may be considered an example of a disjunctive task. If a team of developers is assigned to come up with a suitable (optimal) schema, not all developers are required to come up with the best solution. Instead, it is enough for only one member to provide the best solution. Hence, a highly productive member who is able to come up with the best solution quicker will help the team finish such a task sooner compared to a team whose most productive member has an average or lower productivity rate. Similarly, a task involving the integration of two modules can also be regarded as a disjunctive task. For conjunctive tasks, the overall productivity of a team is defined as the lowest individual productivity. Conjunctive tasks can be considered either divisible focusing on quality, or unitary focusing on quantity. Implementation tasks are an example of software development tasks that can be regarded as conjunctive. For example, in the case where the programming/coding of a module has been split into subtasks for team members to implement individually, the developer possessing the lowest rate of productivity out of thete whole team will determine the team's overall rate of productivity. Compensatory and discretionary types of tasks are not adopted in the approach as they are not considered to be applicable in the case of software development tasks. In compensatory tasks, the overall productivity

of a team can be expressed as the variability of individual productivity. In this attempt though, it does not make sense to require developers to have diversity in their rates of productivity. In discretionary tasks, the decision on how to combine individual contribution is left up to the team members. However, in this approach it is assumed that the output of each software task can only be derived using one specific type.

Finally, each task incurs a communication overhead (t_i^{overhead}) depending on the number of developers assigned to carry out the task. Typically, when developers work together as a group, there is an amount of time spent on communicating with each other in order to coordinate activities, discuss issues and resolve conflicts regarding a task. Communication can take many forms (such as meetings, phone calls, e-mails, video calls, etc.) all of which take away from work that the developers are assigned to perform. The approach proposes, therefore, to take into account this additional time needed for communication by adjusting the make span of each task in order to give a more accurate project duration and cost.

According to Brooks [104], there is a polynomial relationship linking the size of a team working on a task with the number of possible communication paths between pairs of developers. Specifically, the relationship between the number of developers,t_i^{assigned}, assigned to task t_i and the number of communication paths, t_i^{paths}, can be defined using

$$t_i^{\text{paths}} = \frac{t_i^{\text{assigned}} \times (t_i^{\text{assigned}} - 1)}{2}. \tag{7.1}$$

This means that as the number of developers working together on a task increases, so does the number of communication paths, resulting in an exponential growth in communication overhead. Abdel-Hamid and Madnick [105], therefore, carried out an empirical investigation to attempt to quantify the percentage of communication overhead incurred given different team sizes. Their findings are presented in Table 7.1.

In order to determine the percentage of communication overhead for any specific number of developers, Douglas [106] suggests interpolating between the two nearest team sizes given in Table 7.1. By applying linear regression, he was then able to formulate an equation (Equation (7.2)) using the number of communication paths as a variable to calculate the percentage of communication overhead of any team size. This approach adopts the formula to calculate the communication overhead, t_i^{overhead}, of each task t_i given by

$$t_i^{\text{overhead}} = 0.001248269 \times t_i^{\text{paths}}. \tag{7.2}$$

Table 7.1: Correlation between team size, number of communication paths and percentage of communication overhead

Team Size	Communication Paths	Communication Overhead
0	0	0.00%
5	10	1.50%
10	45	6.00%
15	105	13.50%
20	190	24.00%
25	300	37.50%
30	435	54.00%

With the above information regarding tasks and developers, the goal is to allocate developers and schedule tasks in such a way that the shortest possible project make span and cost are achieved simultaneously. It is assumed that a task can be assigned more than one developer, and a developer can be assigned to tasks associated with different professions as long as they possess the required skills and have a positive nonzero productivity rate. Also, a developer can be assigned to work on only one task at any given time. This means that a developer will not be set to work on tasks that are executed concurrently, thus avoiding conflicts in assignment. Furthermore, tasks cannot be preempted, that is, once a task starts it must be completed and its execution cannot be suspended. In the case of divisible tasks, a developer may finish his or her contribution earlier than other team members. Normally, this would allow the developer to be free to work on another task. However, in this approach, it is considered that developers are assigned to a task as a team and so all developers will remain assigned until the whole task is complete and they will be paid for the whole duration of the task. In order to achieve these goals and satisfy the underlying constraints, the objectives and constraint functions are modified in order to guide the generation of (near-)optimal and feasible solutions.

7.2.1 Representation and encoding

Candidate solutions are represented by individuals in the population that are composed of two variables: one to handle the information regarding the allocation of resources and one to handle the information related to the scheduling of tasks, as similarly proposed by Yannibelli and Amandi [77]. An example of the encoding of a candidate solution using the TPG of Figure 4.1 is shown in Figure 7.1. The first variable is encoded using a bitset array of length m, where each element u of the array contains a bitset that represent only those developers that possess the skills of the

profession required to carry out task t_u. If the value of a bit in the sequence is '0', then the corresponding developer is not assigned to the task, whereas if the value of the bit in the sequence is '1', then the corresponding developer is assigned to the task. With this representation, the total number of bits required always varies according to the number of tasks and number of available developers in each profession. The second variable uses a permutation array whose length is also equal to the number of tasks in the project, m. Each element v of the array contains the index of a task in the project. Tasks can only appear once in the array and are chosen for scheduling in order of their appearance from left to right in the array.

u	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
	101	00100	10001	01011	0110100	0010101	1000100	010

v	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
	1	3	2	5	4	6	7	8

Figure 7.1: Example of the encoding used to represent an individual: (top) developer allocation variable, and (bottom) task schedule variable in Approach 4

7.2.2 Population initialization

As with the previous approaches, individuals of the population must be initialized subject to several feasibility restrictions in order to guarantee that the candidate solutions they represent are valid. At the same time, the initialization should ensure that the population represents a varied and heterogeneous pool of candidate solutions. This can be achieved by randomly setting the starting values of the two variables subject to the necessary constraints. For the bitset representing the allocation of developers, each task must have at least one developer assigned to it. For the permutation array representing the scheduling order of tasks, each task must appear in the array after all of its predecessors in order for the solution to be feasible. To ensure this, the initialization follows a process of randomly selecting a task only from a set that consists of tasks that have not yet been selected and whose predecessors have already been inserted in the array. Thus, the initial permutation arrays will always represent feasible project schedules, since no task will be selected before its predecessors.

7.2.3 Optimization function

The optimization adopts two objective functions, $\mathcal{F}_{\text{duration}}(x)$ and $\mathcal{F}_{\text{cost}}(x)$, in order to assess the fitness of each individual solution x in the population. The values given to these two objective functions denote the duration and the cost of the project, respectively. Also, the approach uses two constraint functions, $\mathcal{G}_{\text{assignment}}(x)$ and $\mathcal{G}_{\text{dependency}}(x)$, which assess the feasibility of an individual solution. The first constraint function reflects whether or not at least one developer is assigned to a task, and the second one reflects whether or not any dependency violations between tasks exist. The following subsections describe in detail how the values of each function are calculated.

The goal of the optimization is to consider the productivity-related characteristics of tasks and developers so as to perform allocation and scheduling in a way that simultaneously minimizes project duration and project cost of a solution x of the population, POP, bound by the constraints. At the same time, the two objectives are competing in nature, that is, attempting to decrease one objective would lead to an increase in the other. The optimization will therefore provide a set of (near-)optimal solutions rather than a single (near-)optimal solution. Thus, the goal is to minimize a vector consisting of the two objective functions given as

$$\begin{aligned} \underset{\forall x \in POP}{\text{Minimize}} \quad & \mathcal{F}(x) = (\mathcal{F}_{\text{duration}}(x), \mathcal{F}_{\text{cost}}(x)), \\ \text{subject to} \quad & \mathcal{G}_{\text{assignment}}(x) = 0, \\ & \mathcal{G}_{\text{dependency}}(x) = 0. \end{aligned} \tag{7.3}$$

For each individual x in the population, POP, the two objective functions are evaluated simultaneously. As the algorithm attempts to improve the quality of the population, during selection individuals are compared against each other using both their objective function values to ascertain which individuals are nondominated as previously described in the beginning of Chapter 6. Ultimately, since each solution represents a different allocation and scheduling plan, the project manager will be offered a choice on which plan he or she feels suits the project and the organization better.

7.2.3.1 Objective functions

7.2.3.1.1 Project duration objective function

In order to compute the overall duration of a project, $\mathcal{F}_{\text{duration}}(x)$, represented by a solution x in the population, POP, the approach first calculates the duration of each task individually to

determine the amount of time (in hours) that developers will spend on each task they are assigned to. Then, by using the precedence relationships between tasks and the availability of developers, the starting and finishing times of each task are determined. The project duration is then established as the highest finishing time of all tasks.

The duration of a task is calculated using the productivity rate of developers, similarly to the way presented in Kapur et al. [70] and Ngo-The and Ruhe [71], but also using the type of task interdependence. The order in which task durations are computed is determined by the order that tasks appear in permutation array variable. With this forward scheduling approach, the first element of the permutation array (position $v = 1$) contains the index u of the task whose duration is to be calculated first. Using this index, the corresponding element at position u of the bitset array variable is decoded in order to obtain the set of developers, A^i, assigned to task t_i. Once this set of developers has been determined, the duration is then calculated by dividing the effort that t_i is estimated to require by the overall productivity rate (t_i^{prod}) of the team of developers comprising the set A^i, as shown in Equation (7.4).

$$t_i^{\text{duration}} = \frac{t_i^{\text{effort}}}{t_i^{\text{prod}}}. \tag{7.4}$$

The overall productivity rate t_i^{prod} is computed using Equation (7.5), which takes into account the information in the productivity matrix, $PROD$, concerning each developer $r_j \in A^i$ and the task's interdependence type, t_i^{type}.

$$t_i^{\text{prod}} = \begin{cases} \text{sum}\{prod_{jl} \mid r_j \in A^i\}, & \text{if } t_i^{\text{prof}} = h_l \text{ and } t_i^{\text{type}} = additive \\ \max\{prod_{jl} \mid r_j \in A^i\}, & \text{if } t_i^{\text{prof}} = h_l \text{ and } t_i^{\text{type}} = disjunctive \\ \min\{prod_{jl} \mid r_j \in A^i\}, & \text{if } t_i^{\text{prof}} = h_l \text{ and } t_i^{\text{type}} = conjunctive \end{cases}. \tag{7.5}$$

The term $prod_{jl}$ denotes the rate of productivity possessed by developer r_j at profession h_l. Using the operationalization suggested by Steiner [100], if task t_i is categorized as additive, then the summation operator is used to aggregate individual productivity rates. Alternatively, if task t_i is categorized as disjunctive, then the maximum operator is applied. Else, if task t_i is categorized as conjunctive, then the minimum operator is employed.

With this information, the duration can be adjusted to take into account the communication overhead calculated from Equations 7.1 and 7.2 accordingly, using Equation (7.6) for calculating

the final duration of a task. The ceiling function rounds up to the nearest hour.

$$t_i^{\text{duration}} = \lceil \frac{t_i^{\text{effort}}}{t_i^{\text{prod}}} \times \frac{1}{1 - t_i^{\text{overhead}}} \rceil. \tag{7.6}$$

Following this, the earliest starting time and finishing time of task t_i are then calculated. As shown in Equation (4.2), the earliest starting time, t_i^{start}, of task t_i is determined by the maximum finishing time out of all its predecessor tasks. If task t_i has no predecessor tasks, then it can begin immediately.

$$t_i^{\text{start}} = \begin{cases} 0, & \text{if } \nexists t_j \text{ such that } (t_j, t_i) \in D \\ \max\{t_j^{\text{finish}} \mid (t_j, t_i) \in D\} + 1, & \text{otherwise} \end{cases}. \tag{4.2 revisited}$$

In the previous three attempts, the aim was to minimize the delay between the *scheduled* starting time of a task (encoded in the individuals of the population) and the *earliest* starting time (computed using Equation (4.2)). The finish time of a task, therefore, depended on its scheduled start time (Equation (4.3)). However, in this attempt the information regarding the actual start time of tasks has been removed from the representation and replaced by a permutation array. Consequently, the durations of the tasks are computed in the order in which they appear in the permutation array, meaning that each task will be scheduled at the earliest possible starting time taking into account the finishing times of its predecessor tasks. The finishing time, t_j^{finish}, of a task t_j is simply equal to the earliest start time (as there is no longer a scheduled start time) plus its duration, as shown in Equation (7.7).

$$t_j^{\text{finish}} = t_j^{\text{start}} + t_j^{\text{duration}}. \tag{7.7}$$

However, it is also necessary to examine the availability of the assigned developers in order to avoid any conflicts that will cause a schedule to be infeasible. The procedure carried out to handle this is presented in Algorithm 7.1. First, the start and finish times of task t_i are calculated using Equations 4.2 and 7.7, respectively. If all developers $r_k \in A^i$ are available for the duration, t_i^{duration}, of task t_i, then no modifications to the start and finish times are necessary. However, if at least one of the developers assigned to carry out task t_i is already assigned to a different task between the start time and finish time of task t_i, then the value of t_i^{start} is adjusted to the next time step and t_i^{finish} is recalculated again using Equation (7.7). This process repeats until the earliest time is determined that satisfies that all developers are available to work for the whole duration of the task.

Algorithm 7.1 Procedure to compute earliest start and finish time of a task.

Input: Set of developers A^i assigned to task t_i, start time t_i^{start}, duration t_i^{duration} and finish time t_i^{finish}.

conflict $\leftarrow$ true
while conflict **do**
    for all developers $r_k \in A^i$ **do**
        if developer r_k is assigned to any task t_j at a time unit such that
        $t_i^{\text{start}} \leq t_j^{\text{start}} \leq t_i^{\text{finish}}$ **then**
            $t_i^{\text{start}} = t_i^{\text{start}} + 1$
            $t_i^{\text{finsih}} = t_i^{\text{start}} + t_i^{\text{duration}}$
            conflict $\leftarrow$ true
        else
            conflict $\leftarrow$ false

Output: Start time t_i^{start} and finish time t_i^{finish} of task t_i.

Once all task start and finish times have been determined, then the overall duration of the software project represented by solution x is calculated by taking the value corresponding to the highest finishing time out of all m tasks. This value corresponds to the value given to the first objective function, $\mathcal{F}_{\text{duration}}(x)$, and is defined in Equation (7.8) as

$$\mathcal{F}_{\text{duration}}(x) = \max\{t_i^{\text{finish}} \mid t_i \in T\}. \tag{7.8}$$

7.2.3.1.2 Project cost objective function

To evaluate the overall cost of a software project, $\mathcal{F}_{cost}(x)$, represented by a solution x in the population, POP, the approach begins by calculating how much the assigned developers will cost for each task, and then aggregating all individual task costs. Specifically, the cost, t_i^{cost}, of task t_i is computed by aggregating how much each assigned developer $r_k \in A^i$ will cost for the duration of the task based on his or her wage rate, as given in Equation (7.9).

$$t_i^{\text{cost}} = \sum_{j=1, r_j \in A^i}^{t_i^{\text{n}}} t_i^{\text{duration}} \times r_j^{\text{salary}}. \tag{7.9}$$

Subsequently, the overall cost of developers for the project represented by solution x is computed by summing the cost of all m tasks individually. This value corresponds to the value given to the second objective function, $\mathcal{F}_{\text{cost}}(x)$, and is defined in Equation (7.10) as

$$\mathcal{F}_{\text{cost}}(x) = \sum_{i=1}^{m} t_i^{\text{cost}}. \tag{7.10}$$

7.2.3.2 Constraint functions

The feasibility of each candidate solution x in the population, POP is assessed by again using the information stored in the two variables. Since the approach uses forward scheduling based on the availability of the assigned developers, there is never a violation where a developer is assigned to more than one task at any given time. Hence, the only constraints evaluated concern (a) whether or not a task is assigned at least one developer and (b) whether the scheduling of tasks conforms to the precedence relationships. For the former, the value is equivalent to the number of tasks that have no developers assigned, which is calculated by the conditional summation in the constraint function of Equation (7.11).

$$\mathcal{G}_{\text{assignment}}(x) = \sum_{i=1}^{m} [t_i^{\text{n}} = 0]. \tag{7.11}$$

For the latter, the value equals to the number of dependencies violated by the schedule, which computed by the conditional summation given in the constraint function of Equation (7.12).

$$\mathcal{G}_{\text{dependency}}(x) = \sum_{(t_i,t_j)\in D}^{|D|} [t_j^{\text{start}} \leq t_i^{\text{finish}}]. \tag{7.12}$$

7.2.4 Genetic operators

In this approach, two binary tournaments are performed in order to select the parents to create offspring. In each tournament, a pair of individuals is randomly selected as candidate parents, which are then compared based on their dominance. Specifically, if one candidate parent dominates the other candidate parent (that is, betters the other in at least one of the objective functions, and betters or is equal to the other in the remaining objective functions), then that candidate is declared 'winner' of the tournament, and is chosen as a parent. In this way, individuals with greater fitness have a better chance of becoming parents and surviving into the next generation. If the fitness of two candidates is tied, then the candidate with the lowest number of violations as determined by the constraint functions is chosen. For the crossover operator, the developer allocation variable uses single-point crossover at a random bit of the bitset array, whereas the task scheduling variable uses partially-mapped crossover [107], which guarantees that the constraint of having each task only appear once is satisfied. For the mutation operator, the developer allocation variable uses a bit-flip operator where a randomly selected bit of the bitset is altered from a value of '0' to a value of '1' or vice-versa. For the task scheduling variable, a swap mutation takes place where two positions

in the permutation array are chosen randomly and the indices in those positions are swapped. In this way, the preservation of the validity of an individual is ensured.

7.3 Experiments

The experimental process was set up with two experiments in order to help answer the two previously-defined research questions, $RQ4A$ and $RQ4B$, respectively.

Specifically, in the first experiment the goal was to compare the performance of four well-known variations of multiobjective genetic algorithms, namely, the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [89], the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [108], the Pareto Archived Evolution Strategy (PAES) [109], and the Multi-Objective Cellular algorithm (MOCell) [110]. These algorithms were selected because they have been extensively used as the underlying mechanism to solve related project scheduling optimization problems in the past. In addition, their use would allow similar future research attempts to be compared with this proposed approach.

A dataset ($DS1$) containing 16 synthetic software projects of varying size, both in terms of the total number of tasks involved, as well as the total number of available developers was generated for this experiment, subject to several conditions. To begin with, four distinct software development projects were constructed with varying size of m: (i) 25 tasks, (ii) 50 tasks, (iii) 75 tasks, and (iv) 100 tasks. Each task in a project was randomly assigned to one of nine software development professions shown in Table 7.2, which were identified using the Standard Occupational Classification (SOC) System [111] and the O*NET Resource Center [112].

Table 7.2: List of software development-related professions used in creation of generated software projects for Approach 4

SOC code	Profession
11.3021.00	Computer and information systems managers
15.1121.00	Computer systems analysts
15.1122.00	Information security analysts
15.1131.00	Computer programmers
15.1134.00	Web developers
15.1143.00	Computer network architects
15.1199.01	Software quality assurance engineers and testers
15.1199.02	Computer systems engineers/architects
15.1122.06	Database architects

In addition, the task precedence graph of each project was created in such a way so as no circular dependencies existed among tasks. Also, each task was assigned a task interdependence type (additive, conjunctive or disjunctive). For the sake of experimentation, this assignment was generated randomly. Finally, the tasks in each project were randomly given an estimated effort value. Next, four separate sets of software developers were randomly generated with different sizes for n: (a) 25 developers, (b) 50 developers, (c) 75 developers, and (d) 100 developers. A productivity matrix was then randomly generated containing values in the range of $[0.0, 2.0]$ making sure that each profession had at least two developers with a value greater than zero to guarantee that all tasks will be able to be completed. Additionally, each developer was assigned a salary indicating his or her wage rate per hour. Salaries were randomly generated within scale ranges in order to reflect that developers with higher levels of expertise and proficiency in skills are more likely to possess a higher productivity rate and, subsequently, cost more in a development company. Finally, each of the four software projects (i)–(iv) was paired with each of the four sets of available workforce (a)–(d) to form 16 project instances.

The second experiment assesses the behaviour of the multiobjective genetic algorithm variations with respect to scalability as the number of tasks and number of available developers increase. To answer this question, the experiment made use of the project instances provided by Luna et al. [66, 113], which were intended for use in experiments that adopted the approach presented by Alba and Chicano [64, 114] This dataset ($DS2$) contains randomly generated projects consisting of six different task sizes (16, 32, 64, 128, 256, and 512) each paired with six different sizes of available developers (8, 16, 32, 64, 128 and 256) for a total of 36 project instances. Because of the underlying differences between approaches, several data present in the instances were then adapted to meet the data requirements of this approach. For example, the instances contained data regarding the skills possessed by developers. This had to be transformed into developer professions so that a developer productivity matrix could be randomly generated for each project instance. In addition, task interdependence types were not present in the instances. Therefore, these also had to be randomly generated based on the number of tasks and number of available developers. The effort required for each task, task precedences and salary of developers in all project instances were left as provided in the original dataset.

Furthermore, it is equally important to investigate the implications of the productivity-based attributes in practical software development settings. For this, several real-world projects were also

investigated. The ultimate goal is to provide an approach that accurately reflects both the manner with which these activities are carried out, and also the factors that may influence decisions taken by software project managers in an automated, efficient and less time-consuming way.

The representation scheme, objective functions and constraint functions explained were implemented for all four variations using jMetal 4.3, a Java-based framework for multiobjective optimization [115]. The same parameters and algorithm settings were used for all algorithms throughout all instances, as summarized in Table 7.3.

Table 7.3: Parameters and algorithm settings used in the execution of the variations in Approach 4

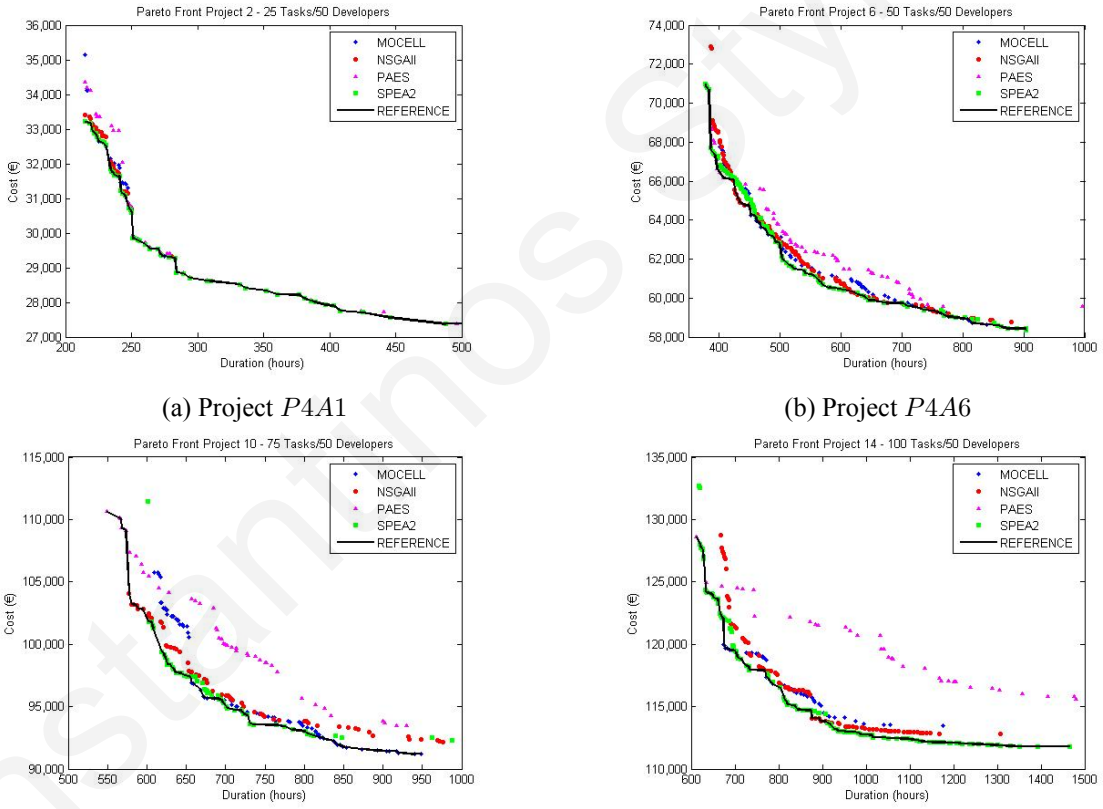
Parameter	Value
Population size	100
Selection operator	Binary tournament
Crossover probability	0.90 (single-point) 0.90 (partiallymapped)
Mutation probability	$1/L$ (bit-flip mutation) 0.90 (swap mutation)
Stopping condition	500,000 objective function evaluations
Number of runs per algorithm	100

Preliminary runs for 50,000 and 100,000 fitness evaluations were performed in order to investigate the convergence of the algorithms with respect to the quality of solutions. Even though the results obtained in these runs showed that the number of fitness evaluations did not actually influence which of the four algorithms performed better, they did show that the quality of solutions could be improved by increasing the number of fitness evaluations. Therefore, in order to allow for a satisfactory trade-off between convergence and computational time, each of the algorithms was executed for 500,000 fitness evaluations.

7.4 Results and discussion

For each project instance, the four algorithms were run 100 times resulting in the generation of 100 final Pareto fronts, each consisting of a number of non-dominated resource allocation and task scheduling solutions. Subsequently, by combining the 100 Pareto fronts, an approximation Pareto front was extracted containing the best solutions each algorithm managed to locate for a project instance over its 100 runs. Then, by combining the approximation Pareto fronts of all four algorithms, a reference Pareto front was identified consisting of the overall best solutions found for each project instance. Consequently, each project instance had four approximation Pareto

fronts (one for each algorithm) and one reference Pareto front (combining the best solutions of all algorithms). Figure 7.2 displays the approximation and reference Pareto fronts achieved for projects instances in dataset $DS1$ with 50 developers and (a) 25 tasks, (b) 50 tasks, (c) 75 tasks and (d) 100 tasks. In the smaller-sized project instances (projects $P4A2$ and $P4A6$), the approximation Pareto fronts of individual algorithms overlap the reference Pareto front, indicating possibly that they are as equally able to find the same (near-)optimal solutions. However, as the size of the projects increases (projects $P4A10$ and $P4A14$), both in terms of tasks and developers, it can be observed that fewer overlaps with the reference Pareto front occur, as well as greater differences in the shape of the individual approximation curves. This could mean that each algorithm is able to locate (near)-optimal solutions in different regions of the solution space. The remaining Pareto fronts are plotted in Figure B.1.



(a) Project $P4A1$

(b) Project $P4A6$

(c) Project $P4A10$

(d) Project $P4A14$

Figure 7.2: Pareto fronts corresponding to the best solutions generated for the first experiment of Approach 4 by the four variations for the project instances in dataset $DS1$ with 50 developers and (a) 25 tasks, (b) 50 tasks, (c) 75 tasks and (d) 100 tasks

The hypervolume [116] and inverted generational distance [91] quality indicators were selected to help compare the four algorithms with respect to performance and scalability given their ability to assess both convergence and diversity (uniformity and spread) of algorithms. Speifically, the hypervolume (HV) indicator assesses the volume covered by the nondominated solutions of a Pareto front in the objective space. Therefore, the larger the volume covered by the solutions generated in a run, then the higher the HV value, which indicates a better performance. Since each algorithm was run 100 times, 100 corresponding HV indicator values were calculated for each algorithm for each project instance. The inverted generational distance (IGD) indicator assesses how far the elements of the true Pareto front are from the nondominated points of an approximation Pareto front. Therefore, the greater the extent of the true Pareto front that is covered by the nondominated points generated by a run in the objective space, the lower the IGD value, which denotes a better performance. In this case, because it is not possible to know the true Pareto front a priori, the reference Pareto front is used instead. Similarly, 100 IGD values were calculated for the 100 runs of each algorithm for each project instance.

7.4.1 RQ4.1: How do different multiobjective genetic algorithm variations perform in terms of generating (near-)optimal solutions with respect to this approach for resource allocation and task scheduling?

In order to compare the performance of the four algorithms, the median HV and IGD values for each algorithm were calculated for each project instance in dataset $DS1$. The values are presented in Tables 7.4 and 7.5, respectively. The shaded cells indicate which algorithm(s) achieved the best value in each project instance (highest median value in the case of the HV indicator and the lowest median value in the case of the IGD indicator). In addition, the average rank of each algorithm is also given.

Using the HV and IGD values obtained, a Friedman rank sum test on both quality indicators (with level of significance $\alpha = 0.05$) was carried out to detect whether or not a statistically significant difference exists among the four algorithms. For the HV indicator, the test produced a Friedman statistic $\chi_F^2 = 35.3846$, p-value: 1.010×10^{-07}, whereas for the IGD indicator the test returned a Friedman statistic $\chi_F^2 = 34.6212$, p-value: 1.464×10^{-07}. For both indicators, the critical chi-square value at $\alpha = 0.05$ for $k - 1 = 3$ degrees of freedom is computed at 7.815,

Table 7.4: Median HV values obtained after 100 runs of each algorithm for the 16 project instances in dataset $DS1$ for the first experiment of Approach 4

Project			**Median HV value**			
Instance	**Tasks**	**Developers**	**MOCell**	**NSGA-II**	**PAES**	**SPEA2**
$P4A1$	25	25	0.9231	0.9231	0.9106	0.9231
$P4A2$	25	50	0.9533	0.9486	0.9333	0.9431
$P4A3$	25	75	0.8028	0.7908	0.7250	0.7585
$P4A4$	25	100	0.8333	0.8438	0.6501	0.7867
$P4A5$	50	25	0.7364	0.7484	0.6286	0.7468
$P4A6$	50	50	0.6206	0.6120	0.4925	0.5886
$P4A7$	50	75	0.6551	0.6733	0.4891	0.6189
$P4A8$	50	100	0.1773	0.1217	0.0000	0.0915
$P4A9$	75	25	0.7942	0.7771	0.6175	0.7687
$P4A10$	75	50	0.3755	0.3848	0.1940	0.3916
$P4A11$	75	75	0.4295	0.4280	0.1431	0.3883
$P4A12$	75	100	0.3951	0.3911	0.1648	0.2575
$P4A13$	100	25	0.5352	0.4754	0.1894	0.4638
$P4A14$	100	50	0.4765	0.4761	0.1391	0.4469
$P4A15$	100	75	0.0841	0.0842	0.0000	0.0701
$P4A16$	100	100	0.0935	0.0986	0.0000	0.1388
Average ranking			**(1) 1.6250**	**(2) 1.7500**	**(3) 4.0000**	**(4) 2.6250**

Table 7.5: Median IGD values obtained after 100 runs of each algorithm for the 16 project instances in dataset $DS1$ for first experiment of Approach 4

Project			**Median IGD value**			
Instance	**Tasks**	**Developers**	**MOCell**	**NSGA-II**	**PAES**	**SPEA2**
$P4A1$	25	25	0.0001	0.0001	0.0007	0.0001
$P4A2$	25	50	0.0014	0.0014	0.0015	0.0014
$P4A3$	25	75	0.0066	0.0073	0.0111	0.0086
$P4A4$	25	100	0.0144	0.0153	0.0185	0.0154
$P4A5$	50	25	0.0009	0.0009	0.0021	0.0009
$P4A6$	50	50	0.0128	0.0127	0.0170	0.0144
$P4A7$	50	75	0.0160	0.0150	0.0272	0.0177
$P4A8$	50	100	0.0536	0.0597	0.0859	0.0638
$P4A9$	75	25	0.0081	0.0082	0.0124	0.0080
$P4A10$	75	50	0.0280	0.0272	0.0418	0.0265
$P4A11$	75	75	0.0301	0.0304	0.0572	0.0330
$P4A12$	75	100	0.0326	0.0327	0.0498	0.0412
$P4A13$	100	25	0.0189	0.0220	0.0435	0.0225
$P4A14$	100	50	0.0215	0.0216	0.0432	0.0229
$P4A15$	100	75	0.0398	0.0399	0.0951	0.0413
$P4A16$	100	100	0.0385	0.0382	0.0666	0.0345
Average ranking			**(1) 1.6250**	**(2) 1.9375**	**(3) 4.0000**	**(4) 2.4375**

which is lower than the respective statistics. Hence the tests led to the rejection of the null hypothesis that the algorithms are equivalent with respect to both the HV and IGD indicators. Since the Freidman tests strongly suggested that significant differences do exist between at least two algorithms, a multiple pairwise comparison of algorithms was carried out to identify exact differences between pairs of algorithms. To handle the family-wise error rate accumulated, p-values were adjusted using a post-hoc Holm procedure. The results of the comparison are shown in Table 7.6, where pairs of algorithms with a statistically significant difference ($p < 0.05$) are shown shaded. According to the pairwise comparisons, no significant difference is observed between MOCell and NSGA-II, MOCell and SPEA2, and NSGA-II and SPEA2 in either indicator. However, MOCell, NSGA-II and SPEA2 all have statistically significant differences with PAES. Since the HV and IGD indicators relate to the convergence and diversity of a Pareto front, the approximation Pareto fronts generated by MOCell, NSGA-II and SPEA2 can be considered to cover a larger volume of the objective space and are nearer to the (near-)optimal compared to PAES. Therefore, to answer RQ4.1, MOCell, NSGA-II and SPEA2 would perform better as the underlying multiobjective optimization mechanism for this approach since, based on the statistical analysis, they are equally capable of generating a more diverse range of trade-offs between project duration and project cost that correspond to resource allocations and task schedules.

Table 7.6: Adjusted p-values resulting from the pairwise comparison ($\alpha = 0.05$) for HV and IGD indicators in first experiment of Approach 4

	HV adjusted p -values		
	NSGA-II	**PAES**	**SPEA2**
MOCell	0.784191	0.000001	0.085379
NSGA-II	–	0.000004	0.110468
PAES	–	–	0.010365

	IGD adjusted p -values		
	NSGA-II	**PAES**	**SPEA2**
MOCell	0.546643	0.000001	0.225180
NSGA-II	–	0.000031	0.546643
PAES	–	–	0.002475

7.4.2 RQ4.2: How do different multiobjective genetic algorithm variations behave in terms of scalability as the number of tasks and developers increases in this approach for resource allocation and task scheduling?

In order to compare the scalability of the algorithms, the approach was examined with the 36 instances found in dataset $DS2$, and followed the method described in Luna et al. [66]. Scalability was assessed in terms of number of tasks and available developers separately, again using the HV indicator as a basis of comparison due to the fact that this metric considers the diversity of solutions and also the convergence of algorithms. In the same way as described previously, the median HV value over 100 runs of each algorithm was calculated for all 36 instances.

First, the algorithms were assessed regarding how they behave as the size of the projects increases in terms of number of tasks. To begin with, project instances were grouped together based on the number of tasks they contained. This led to six groups of project instances. Then, for each algorithm, the average of each group's median HV values were calculated. Table 7.7 shows these averages for all six different task sizes in dataset $DS2$. The results of the table are also shown graphically in the bar chart of Figure 7.3.

Table 7.7: Average median HV values per algorithm for project instances with the same number of tasks over all sizes of available developers obtained for second experiment of Approach 4

	Average median HV values			
Task size	**MOCell**	**NSGA-II**	**PAES**	**SPEA2**
16	0.4130	0.4190	0.3859	0.4099
32	0.2692	0.2765	0.2262	0.2687
64	0.0846	0.0805	0.0000	0.0763
128	0.0681	0.0946	0.1384	0.0893
256	0.0037	0.0000	0.1799	0.0000
512	0.0000	0.0000	0.3151	0.0000

It is generally expected that the higher the number of tasks, the harder it will be for the algorithms to find (near-)optimal solutions. Indeed, this does hold true in here, as it is observed that as the number of tasks increases from 16 to 64, the averaged HV values tend to worsen for all algorithms with a steep slope. For MOCell, NSGA-II and SPEA2, the increase between 64 and 128 tasks shows a steady behaviour of the algorithms with respect to scalability as there is little change in the averaged HV values. Interestingly, as the number of tasks increase from 128 to 512, the averaged HV value actually increases for PAES.

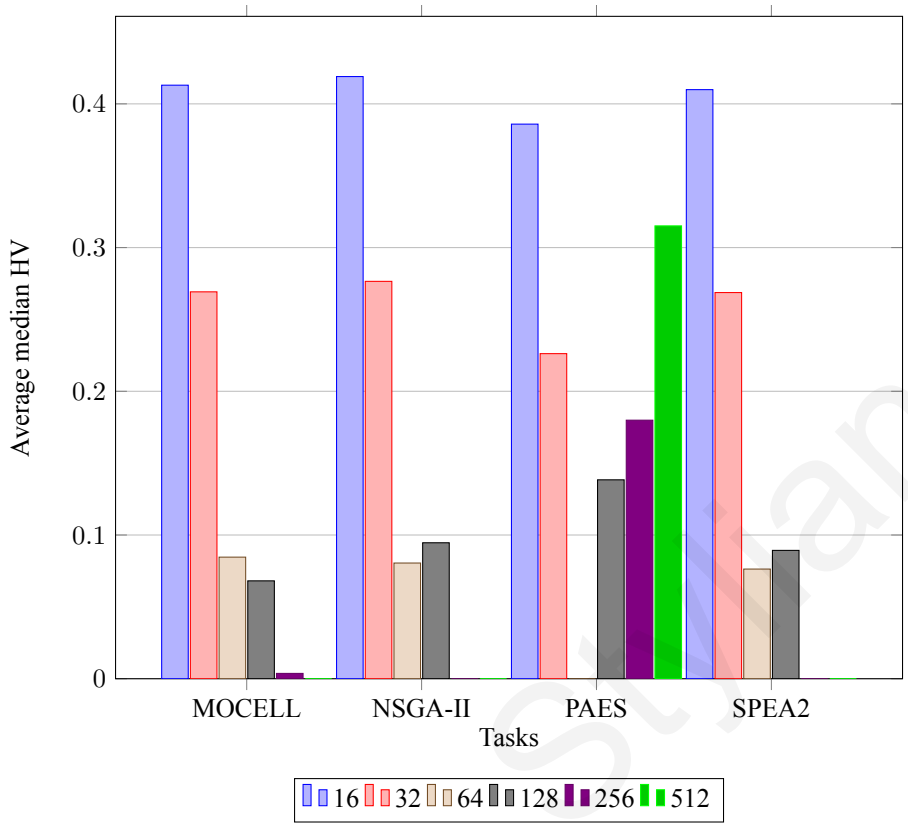


Figure 7.3: Comparison of average HV values per algorithm for instances with the same number of tasks over all sizes of developers obtained for second experiment of Approach 4

In a similar fashion, the algorithms were evaluated as to how they behave as the size of the projects increases in terms of number of available developers. Again, instances of projects were grouped, but this time by those having the same number of developers. Then, for each algorithm, the HV values of each group were averaged. The averages for all six developer sizes of dataset $DS2$ are presented in Table 7.8 and the equivalent bar graph is shown in the bar chart of Figure 7.4.

Table 7.8: Average median HV values per algorithm for project instances with the same number of developers over all sizes of available tasks obtained for second experiment of Approach 4

	Average median HV values			
Developer size	**MOCell**	**NSGA-II**	**PAES**	**SPEA2**
8	0.4793	0.5017	0.3582	0.4924
16	0.2631	0.2714	0.2159	0.2636
32	0.0739	0.0619	0.0938	0.0594
64	0.0222	0.0356	0.1584	0.0289
128	0.0001	0.0000	0.2126	0.0000
256	0.0000	0.0000	0.2066	0.0000

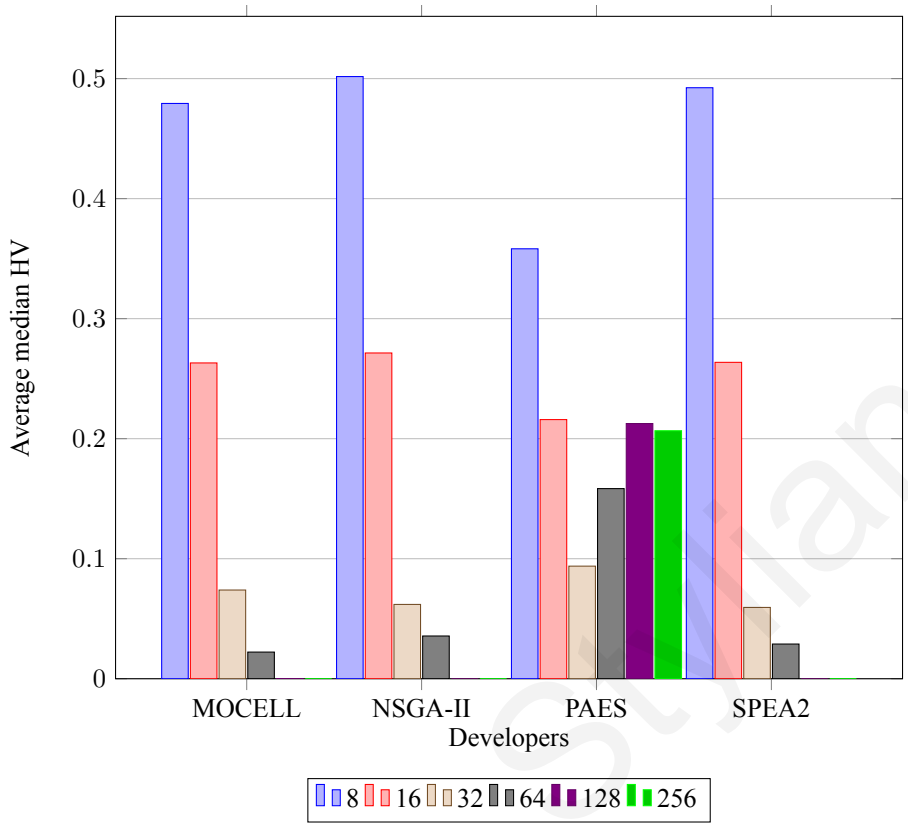


Figure 7.4: Comparison of average HV values per algorithm for instances with the same number of developers over all sizes of tasks obtained for second experiment of Approach 4

Once more, it is generally expected that as the number of developers increase, the HV values will decrease. This is mirrored in the behaviour of the MOCell, NSGA-II and SPEA2 algorithms, which show that they do not scale considerably well, but rather have a sharp gradient. On the other hand, the averaged HV values for PAES show that the algorithm exhibits a better ability for scaling. Overall, the results indicate that PAES is superior with respect to scalability of both task size and developer size, which is similarly concluded in Luna et al. [66, 113].

7.4.3 RQ4.3: What observations can be made from the application of the approach in real-world software projects?

Several software projects carried out by university students were examined in order to assess how productivity rates and interdependence type actual impact software development. In particular, these projects were undertaken by third-year university students during a project-oriented

Software Engineering course spanning two semesters. The students possessed software development knowledge and skills at different levels of proficiency, and also had gained various practical experiences from their university coursework and, in some cases, industry employment. Groups of four to five students were required to develop a software product following a traditional waterfall life-cycle model and plan-driven development based on requirements from external real-world clients. In some cases, group members worked on tasks individually, whereas in other cases, tasks required two or more group members to work together as a team.

Considering that the projects were approximately of the same size, difficulty and complexity, several comparisons could be made regarding how the characteristics of the tasks and the productivity rates of the students influenced the duration of tasks. First, in tasks with a conjunctive interdependence (such as various programming tasks) if at least one student in a team had a relatively low productivity rate then the whole team would struggle and take longer than planned to complete such a task. Second, in tasks with a disjunctive interdependence (such as database designing) even if the majority of the members were not familiar with the task's content, as long as there was one member who possessed the necessary skills and had a high productivity rate, that member was able to help the whole team finish the task sooner. Last, in additive tasks (such as the execution of test cases in the testing phase) students working together with more knowledge in testing had a higher combined productivity rate and, as a result, took shorter time to complete these types of tasks in contrast to teams whose overall team productivity rate was lower. These observations help validate the applicability of this approach that indeed developers combine their efforts in different ways depending on the type of work that needs to be done and their rate of productivity, which subsequently affects task completion times.

To investigate the behaviour of the approach in real-world software development settings, an additional experiment was conducted using a real-world software project undertaken by a local IT company concerning the development of a vessel policies management system for a large insurance brokers company. The supervising project manager at the time of the project had just over five years of industry experience in software project management, and was responsible for the initial planning at the beginning of the project, aiming to find a balance between the total duration and cost of the project. Table 7.9 presents the characteristics of the project, which comprised 31 tasks split into a number of software development activities (professions). The table also shows the interdependence type of each task, which the project manager helped define according to the nature of the activities

in the project. The resources available to undertake the project (presented in Table 7.10) consisted of four developers with skills and expertise in one or more of the professions required by the project tasks. The project manager was consulted to provide the estimated effort for each task, as well as the productivity matrix and salary of the developers. Finally, he also provided the Gantt chart he constructed for the project (Figure 7.5).

Table 7.9: Task characteristics of real-world software project examined in Approach 4

Task	Effort	Type	Profession	Task	Effort	Type	Profession
$T01$	48	Disjunctive	Req. analysis	$T16$	8	Disjunctive	Testing
$T02$	16	Conjunctive	GUI design	$T17$	12	Additive	Programming
$T03$	8	Conjunctive	DB design	$T18$	4	Conjunctive	DB design
$T04$	8	Conjunctive	DB design	$T19$	6	Conjunctive	Programming
$T05$	6	Additive	Req. analysis	$T20$	6	Conjunctive	Programming
$T06$	8	Disjunctive	Testing	$T21$	64	Conjunctive	Integration
$T07$	16	Additive	Programming	$T22$	16	Disjunctive	Testing
$T08$	4	Conjunctive	Programming	$T23$	16	Additive	Programming
$T09$	4	Conjunctive	Programming	$T24$	4	Conjunctive	Programming
$T10$	4	Conjunctive	Programming	$T25$	16	Conjunctive	Programming
$T11$	4	Conjunctive	Programming	$T26$	12	Conjunctive	Programming
$T12$	4	Conjunctive	Programming	$T27$	6	Conjunctive	DB design
$T13$	6	Conjunctive	Programming	$T28$	32	Conjunctive	Integration
$T14$	6	Conjunctive	Programming	$T29$	8	Disjunctive	Deployment
$T15$	8	Conjunctive	Programming	$T30$	12	Additive	Programming
				$T31$	24	Additive	Training

Table 7.10: Characteristics of resources available to undertake the real-world software project examined in Approach 4

	Developer			
	$R01$	$R02$	$R03$	$R04$
Wage rate	€10.23	€6.25	€7.39	€5.68
Productivity rate				
Req. analysis	2.00	0.50	0.50	0.00
DB design	2.00	1.00	2.00	0.00
GUI design	2.00	0.00	2.00	0.00
Programming	2.00	1.00	1.00	0.00
Integration	2.00	0.50	0.50	0.00
Testing	0.00	0.00	0.00	2.00
Deployment	0.00	0.00	0.00	1.00
Training	2.00	0.50	1.00	0.00

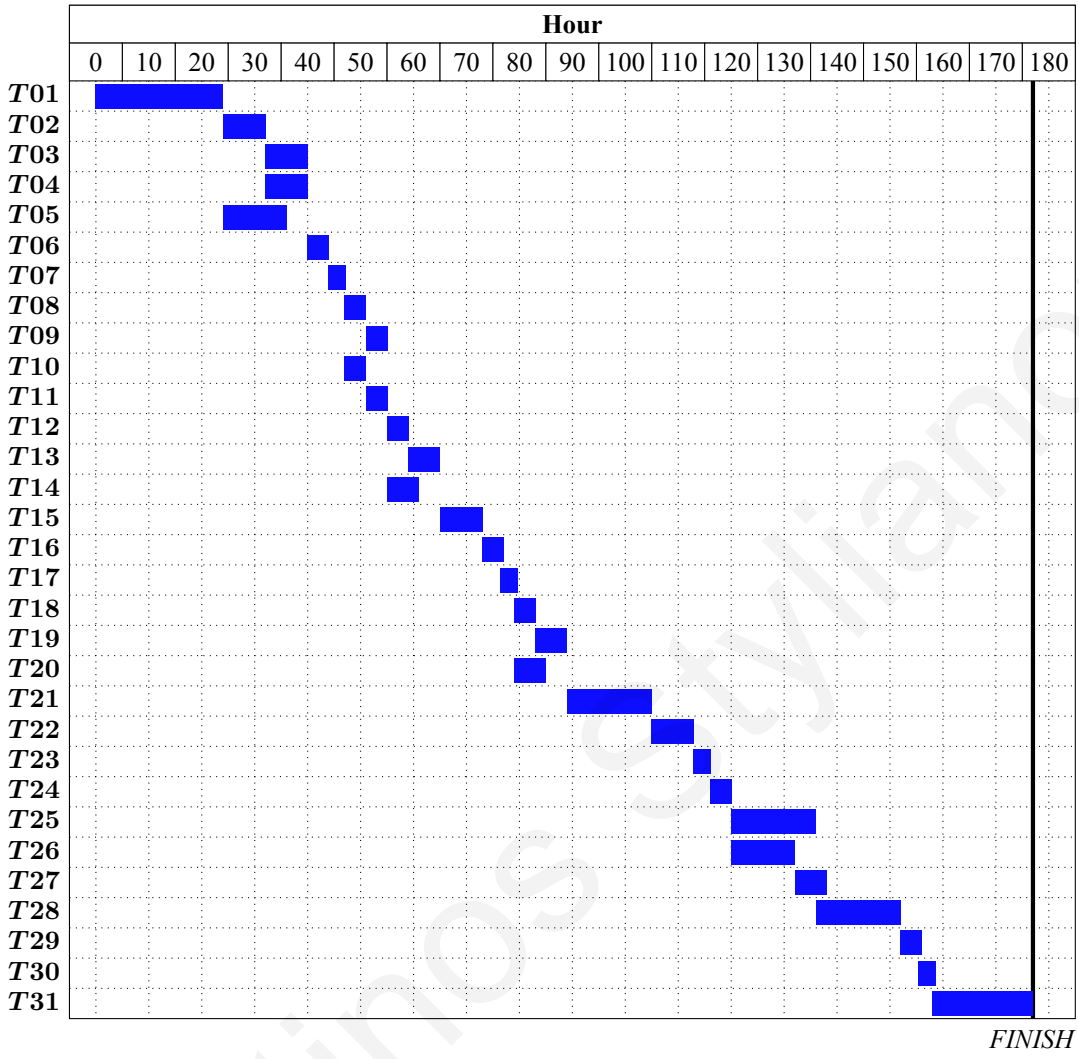


Figure 7.5: Gantt chart constructed by the project manager of the real-world software project examined in Approach 4

Using the same parameters and settings as before, the algorithm was executed for 100 runs for all four algorithms. Subsequently, the HV quality indicator was computed from the solutions generated. The results obtained followed to a large degree the same pattern that was observed with the previous experimental software projects. Specifically, MOCell, NSGA-II and SPEA2 managed to outperform PAES with respect to the HV, demonstrating their ability to generate solutions with higher diversity and cover the extent of the reference Pareto front to a larger degree. Furthermore, there was no statistically significant difference between the results obtained for MOCell and NSGA-II, MOCell and SPEA2, and NSGA-II and SPEA2, which suggests once more that these optimizers are equally suitable for this approach. The reference Pareto containing the overall best

solutions is displayed in Figure 7.6. Also, in the same figure, we plot the duration and cost of the project corresponding to the original allocation of resources and schedule of tasks constructed by the project manager.

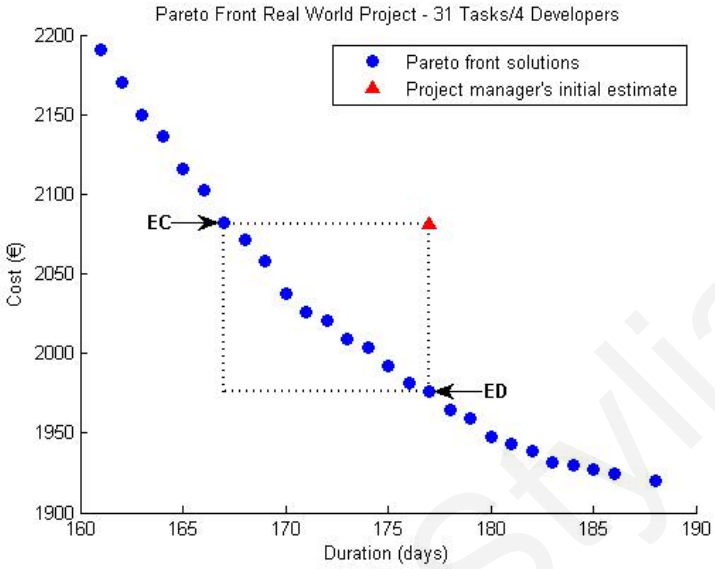


Figure 7.6: Comparison of project manager's initial estimate with reference Pareto front for real-world software project in Approach 4

The 27 solutions on the reference Pareto front all dominate the initial estimate made by the project manager either in terms of duration or in terms of cost. The solutions correspond to project plans ranging from short make spans at higher costs to low costs with longer make span. The project manager's goal was to allocate developers and schedule tasks so that a balance between the total duration and cost was achieved. Notably, the closest generated solutions to the project manager's estimate represent plans that also offer a more equal trade-off between the two objectives (enclosed in the dotted rectangle in Figure 7.6). In other words, the project manager's estimate is not nearer either extreme, but rather closer to the midway solutions. The point on the Pareto front in Figure 7.6 marked ED represents a solution whose task schedule and resource allocation yield a duration equal to the project manager's initial duration estimate. On the other hand, the point EC on the Pareto front represents a solution whose task schedule and resource allocation produce a cost equal to the project manager's initial cost estimate. The comparison of the resource allocations, costs and duration of these two points with the project manager's initial estimate for each task is presented in Table 7.11.

Table 7.11: Comparison of project manager's initial estimation against generated solutions ED and EC in real-world software project examined in Approach 4

	PM's initial estimation			**Generated solution ED**			**Generated solution EC**		
Task	**Resource**	**Duration**	**Cost**	**Resource**	**Duration**	**Cost**	**Resource**	**Duration**	**Cost**
$\boldsymbol{T01}$	$R01$	24	245.52	$R01$	24	245.52	$R01$	24	245.52
$\boldsymbol{T02}$	$R03$	8	59.12	$R01$	8	81.84	$R01$	8	81.84
$\boldsymbol{T03}$	$R03$	8	59.12	$R03$	4	29.56	$R01$	4	40.92
$\boldsymbol{T04}$	$R01$	8	81.84	$R01$	4	40.92	$R03$	4	29.56
$\boldsymbol{T05}$	$R02$	12	75.00	$R01$	3	30.69	$R02$	12	75.00
$\boldsymbol{T06}$	$R04$	4	22.72	$R04$	4	22.72	$R04$	4	22.72
$\boldsymbol{T07}$	$R02$,$R03$	3	40.92	$R01$,$R02$	6	98.88	$R01$,$R02$	6	98.88
$\boldsymbol{T08}$	$R02$	4	25.00	$R02$	4	25.00	$R02$	4	25.00
$\boldsymbol{T09}$	$R02$	4	25.00	$R01$	2	20.46	$R01$	2	20.46
$\boldsymbol{T10}$	$R03$	4	29.56	$R02$	4	25.00	$R03$	4	29.56
$\boldsymbol{T11}$	$R03$	4	29.56	$R03$	4	29.56	$R03$	4	29.56
$\boldsymbol{T12}$	$R02$	4	25.00	$R03$	4	29.56	$R02$	4	25.00
$\boldsymbol{T13}$	$R02$	6	37.50	$R01$	3	30.69	$R01$	3	30.69
$\boldsymbol{T14}$	$R03$	6	44.34	$R01$	3	30.69	$R01$	3	30.69
$\boldsymbol{T15}$	$R03$	8	59.12	$R01$	4	40.92	$R01$	4	40.92
$\boldsymbol{T16}$	$R04$	4	22.72	$R04$	4	22.72	$R04$	4	22.72
$\boldsymbol{T17}$	$R01$,$R02$,$R03$	2	47.74	$R01$	6	61.38	$R01$	6	61.38
$\boldsymbol{T18}$	$R02$	4	25.00	$R03$	2	14.78	$R03$	2	14.78
$\boldsymbol{T19}$	$R02$	6	37.50	$R01$	3	30.69	$R01$	3	30.69
$\boldsymbol{T20}$	$R03$	6	44.34	$R01$	3	30.69	$R01$	3	30.69
$\boldsymbol{T21}$	$R01$,$R02$	16	263.68	$R01$	32	327.36	$R01$	32	327.36
$\boldsymbol{T22}$	$R04$	8	45.44	$R04$	8	45.44	$R04$	8	45.44
$\boldsymbol{T23}$	$R03$	3	22.17	$R01$	8	81.84	$R01$,$R02$	6	98.88
$\boldsymbol{T24}$	$R02$	4	25.00	$R01$	2	20.46	$R01$	2	20.46
$\boldsymbol{T25}$	$R03$	16	118.24	$R01$	8	81.84	$R01$	8	81.84
$\boldsymbol{T26}$	$R02$	12	75.00	$R01$	6	61.38	$R01$	6	61.38
$\boldsymbol{T27}$	$R02$	6	37.50	$R03$	3	22.17	$R03$	3	22.17
$\boldsymbol{T28}$	$R01$	16	163.68	$R01$	16	163.68	$R01$	16	163.68
$\boldsymbol{T29}$	$R04$	4	22.72	$R04$	8	45.44	$R04$	8	45.44
$\boldsymbol{T30}$	$R02$	2	12.50	$R01$	6	61.38	$R01$	6	61.38
$\boldsymbol{T31}$	$R02$,$R03$	19	259.16	$R01$	12	122.76	$R01$,$R02$,$R03$	7	167.09
Project duration		177			177			167	
Project cost			2,081.71			1,976.02			2,081.70

It is clear from Table 7.11 that by taking into account the rate of productivity of developers, the type of task interdependence and the communication overhead at task level, the optimization approach manages to allocate resources and schedule tasks in a variety of ways. The generated solutions can easily be presented to the project manager through a decision support system, from which a project manager may select the most suitable allocation and schedule scheme according to his or her priorities. Furthermore, a project manager is able to generate more accurate solutions

compared to ad-hoc and manual approaches with a small amount of effort. The differences between solutions are also evident when comparing each of the generated solutions with the project manager's initial estimation in regards to the overall project cost and project duration, as shown in Table 7.12.

Table 7.12: Difference between project manager's cost and duration estimation and each generated solution

	Project duration	Difference from estimated duration	Project cost	Difference from estimated cost
PM estimation	177		2,081.71	
Solution				
$RW01$	161	-16	2,190.84	109.84
$RW02$	162	-15	2,170.37	89.37
$RW03$	163	-14	2,149.90	68.90
$RW04$	164	-13	2,136.27	55.27
$RW05$	165	-12	2,115.80	34.80
$RW06$	166	-11	2,102.17	21.17
$RW07$ (EC)	167	-10	2,081.70	0.01
$RW08$	168	-9	2,071.49	-9.51
$RW09$	169	-8	2,057.86	-23.14
$RW10$	170	-7	2,037.39	-43.61
$RW11$	171	-6	2,026.03	-54.97
$RW12$	172	-5	2,020.35	-60.65
$RW13$	173	-4	2,008.99	-72.01
$RW14$	174	-3	2,003.31	-77.69
$RW15$	175	-2	1,991.95	-89.05
$RW16$	176	-1	1,981.70	-99.30
$RW17$ (ED)	177	0	1,976.02	-104.98
$RW18$	178	1	1,964.66	-116.34
$RW19$	179	2	1,958.98	-122.02
$RW20$	180	3	1,947.62	-133.38
$RW21$	181	4	1,943.07	-137.93
$RW22$	182	5	1,938.52	-142.48
$RW23$	183	6	1,931.69	-149.31
$RW24$	184	7	1,929.42	-151.58
$RW25$	185	8	1,927.15	-153.85
$RW26$	186	9	1,924.88	-156.12
$RW27$	188	11	1,920.34	-160.66

As seen from the solutions that are shaded in Table 7.12 (that is, the solutions enclosed in the dotted rectangle in Figure 7.6), compared to the project manager's estimate, the algorithm was able to find a range of alternative plans that are up to approximately 6% shorter in duration for the same estimated cost or up to roughly 5% cheaper in cost for the same estimated duration. The Gantt chart of the latter case (solution ED) is given in Figure 7.7, where the algorithm generated

a solution that managed to allocate resources and schedule tasks with the same project duration as the project manager' estimates, but with a cheaper project cost.

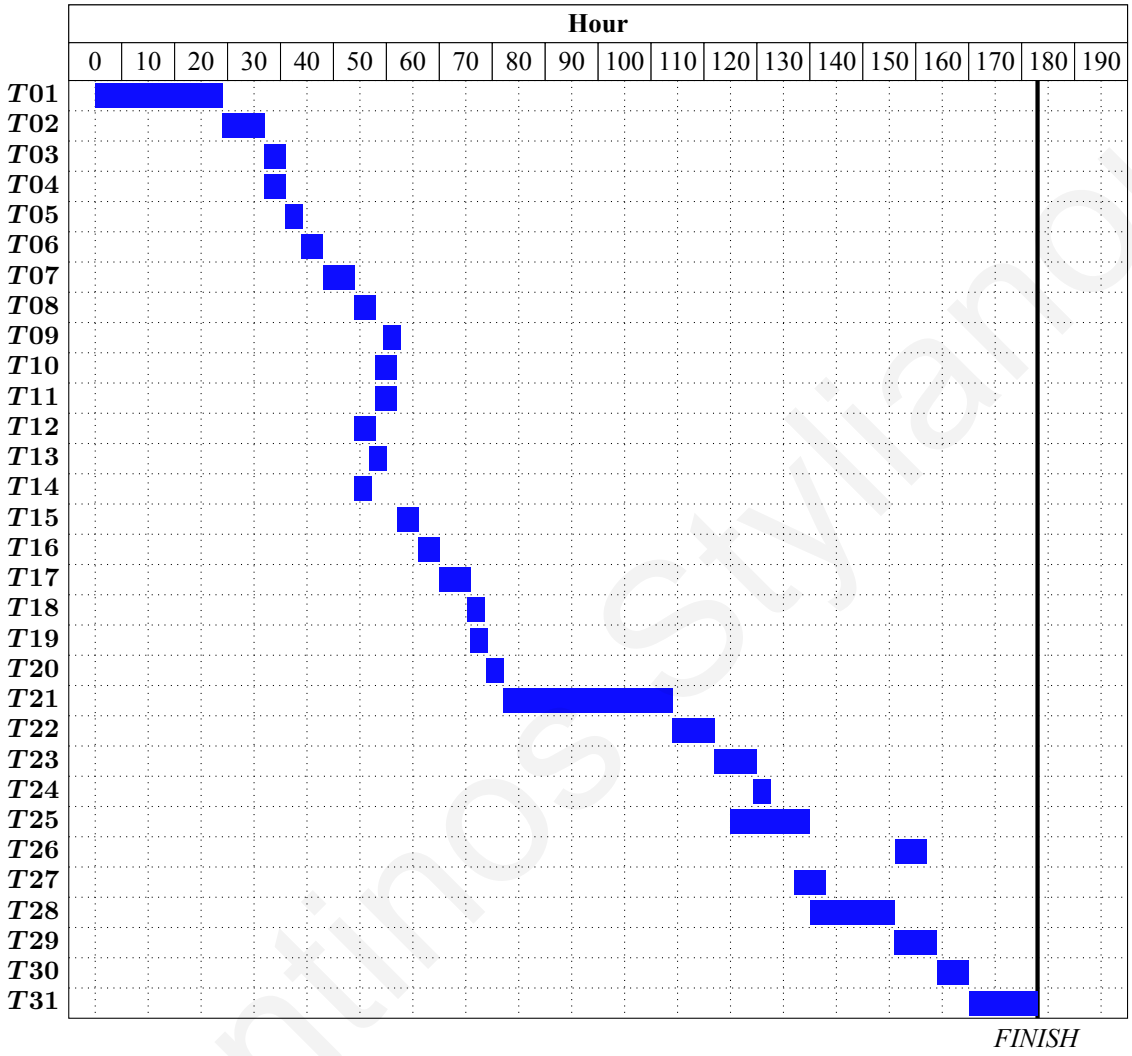


Figure 7.7: Gantt chart corresponding to solution ED generated for the real-world software project in Approach 4

Essentially, the difference in cost is due to having different combinations of developer assignments that are more cost-effective. In some tasks, the developers assigned possessed a high productivity rate and, although these developers cost more, it was still cheaper than assigning developers with a low productivity rate for a longer duration. In other tasks, the developers assigned possessed a low productivity rate and, despite the tasks taking longer, it was still cheaper than assigning developers with a high productivity rate for a shorter duration. This emphasizes

the competitive nature of duration and cost, and underlines the fact that search-based optimization techniques may handle the complexity posed by this competition more efficiently compared to software project managers, irrespective of their experience and expertise. Furthermore, when several of these solutions were presented to the supervising project manager, he confirmed that in several tasks certain combinations of developers that existed both in the original allocation, as well as in the generated solutions completed the work earlier than planned, due to some of the assigned developers possessing high productivity rates in the required professions. In addition, he also confirmed that the final cost and duration of the project was much more near to the solutions generated in this approach rather than his initial estimated values. Although the percentage of reduction was relatively small due to the small size of the project, in larger and more complex projects improvements to the cost and duration estimates could be greater, proving more beneficial to software development companies.

Several observations were also made regarding the interdependence type of tasks. First, the solutions generated by this approach avoided the assignment of more than one developer to certain tasks. Further examination revealed that these tasks were programming tasks, which have a conjunctive interdependence type. Because the duration of a conjunctive task is determined by the lowest productivity rate in the team, the algorithm rejected the assignment of a team of developers to such a task in favour of assigning only one developer that possessed a high rate of productivity, thus decreasing both the duration and cost of the task. Furthermore, due to the dependencies, a number of programming tasks were able to be scheduled to start at the same time (for example, tasks $T08$, $T12$ and $T14$ in Figure 7.7). In order to simultaneously avoid assignment conflicts and minimize the project's make span, the most cost-effective solution was to assign a developer with a lower productivity rate and having the task start as soon as possible (rather than to assign one of the developers with the highest rate of productivity and forcing the start of a task to be delayed until one of those developers was available). Conversely, it was observed that the solutions generated by this approach showed no preference to the number of developers assigned to additive tasks. In some cases, only one developer was assigned (the cheaper alternative), whereas in other cases, several developers were assigned (the faster alternative). These observations suggest that taking into account the type of task interdependence can prove valuable to a software project manager for his/her allocation decisions.

7.5 Summary

The approach described in this chapter has introduced additional aspects for resource allocation and task scheduling that were previously missing from the first three approaches presented in Chapters 4–6. To begin with, the representation of individuals was revised in order to help solve the previous problem of unnecessary delays between the execution of tasks. The new representation allows the tasks to be scheduled exactly at their earliest starting time, thus avoiding idle gaps in the schedule. Also, the levels of experience of developers in skills was replaced by productivity rates of developers in software professions. These productivity rates are now used together with the effort required and the interdependence type of tasks in order to calculate more accurate and realistic task durations before proceeding to allocate resources and schedule tasks. Communication overhead is also added to the duration of tasks to take into account extra time not considered for collaboration between team members in the required effort. In addition, the salary of developers was introduced as a resource attribute in order to calculate the cost of the project using a newly-defined objective function. Through the experiments performed with multiobjective genetic algorithm variations, overall, the approach shows promising results. Applying the approach on a real-world software project demonstrated the importance of incorporating these necessary characteristics of tasks and attributes of developers, which have been either traditionally overlooked or not incorporated fully in previous approaches.

Chapter 8

Personality Psychology and Software Development

Given that a large percentage of software projects are severely challenged or considered to have failed, coupled with the fact that human resources are considered the only type of resource in a software project, one of the directions that the software engineering research community is trying to establish for the software development industry to follow involves including human-centric factors. Specifically, this direction attempts to promote nontechnical factors of software development as equally important as technical factors in approaches for human resource allocation, scheduling and team formation.

A leading trend focuses on taking into account the personality types of developers when assigning them to tasks and also when grouping them into development teams. More and more studies are now being performed aiming to observe the effects of personality types on performance, productivity, software quality, and job satisfaction. Also, there have been attempts to determine the personality types required of different software development professionals, such as system analysts, programmers, testers, etc. Therefore, this chapter presents a literature review of various human resource allocation and scheduling approaches, as well as team formation strategies for software development teams, that incorporate personality types and results of their application in the software industry. Before doing so, several sections are dedicated to presenting an overview of personality testing, including when and how the practice came to be, reasons for employee personality assessment, as well as a brief description of some popular personality testing instruments.

8.1 Background theory of personality psychology

Psychology is the scientific discipline in which human behaviour and mental functions are studied. Personality psychology is the area of psychology that examines the human individual, that is, the person. Though there is no general consensus as to the actual definition of personality, many people incorrectly assume that personality psychology deals with studying individual differences. Every individual has a set of characteristics, both organized and dynamic, that come into action or that are expressed in certain situations regarding a person's cognitive, motivational and behavioural patterns. Over the years many categories of personality theories have been developed, including trait theories, type theories, humanistic theories and behaviourist theories, all of which aim either to understand an individual's distinctive personality features or to identify general rules applying to different individuals. During the 20th century, with the rapid growth of the field of personality psychology, there was an equal interest in the field of personality testing. The intensive research in the field has led to many additions and modifications of personality assessment instruments, both in approach and application.

8.1.1 Humours and temperaments

One of the most important and pioneering figures in the field of medicine is undoubtedly the ancient Greek physician Hippocrates (ca.460–370 BCE). He is known as the father of modern medicine and is responsible for introducing the concepts of crisis and prognosis. He is also credited for developing the concept of humourism, which is based on the idea that an individual's behaviours and personality characteristics are influenced by the four fluids, known as the 'four humours', present in the human body: blood, phlegm, black bile and yellow bile. The word 'humour' originates from the Greek word 'χυμός', meaning 'juice' or 'fluid', which is the reasoning behind the name given to the theory. The four humours/bodily fluids are all present in a human at different levels. A person is considered healthy when there is an even balance of these four humours, but an imbalance in either one would affect both the physical health and personality of the person. Hence, these four humours were presumed to influence the temperament of individuals: sanguine (blood), phlegmatic (phlegm), melancholic (black bile), and choleric (yellow bile). For example, a person with an excess of black bile was said to have a melancholic disposition causing sleeplessness and depression. The doctrine of the four temperaments remained a popular medical theory for centuries until cellular pathology emerged.

8.2 History of employee personality assessment

During the 20th century, with the rapid growth of the field psychology, there was an equal interest in the field of employee personality testing [117]. In 1911, Frederick Winslow Taylor [118] introduced the idea that worker productivity can be increased by applying scientific methods in employee management. He published his work in *The Principles of Scientific Management*, in which he states that traditional knowledge of a task should be gathered and documented from a worker through a series time-and-motion studies in order to describe what needs to be done, how it needs to be done, and how long it should take for it to be completed. He also states that a task should be undertaken by a worker who has the proper skills and has been trained to perform the task in the best way and that this will lead the worker to achieve maximum efficiency. The notions of specialization and division of labour by simplifying jobs and optimizing the way they are executed rely on the fact that employee skills are quantifiable.

Two years later in 1913, the German-American psychologist Hugo Münsterberg published his theories and research work in a book titled *Psychology and Industrial Efficiency* [119], which focused on the field of industrial/organizational psychology. In his work, Münsterberg investigated the desirable psychological abilities and traits an employee was required to have. This test was put into practice by the Boston Elevated Company for selecting conductors, as well as by the American Tobacco Company for choosing travelling salesmen. His works eventually lead to the widespread development and adoption of the use of personality tests to measure and evaluate candidate employees. In addition to the use of screening procedures, other techniques have been developed to include employee character evaluation. For example, when the USA entered World War I, the War Department observed that some recruits were prone to stress and being shell-shocked during battle. As a result, the USA Army hired Robert Sessions Woodworth, an experimental psychologist, to develop a checklist to evaluate soldiers for special missions by carrying out tests for their selection. Despite the fact that the measure was completed in 1919, after the war ended, the Woodworth Personal Data Sheet (WPDS) is the one of the forerunners of modern personality testing put into practice [120].

Personality testing continued to be practised, though after World War I testing began to dwindle as the evidence that job success was directly correlated to personality aptitudes slowly shrunk. Henry Charles Link's *Employment Psychology* published in 1919 regarding industrial hiring states

that ‘this [the weakness of the method of employing] is due to the absolute reliance upon the ability and experience of the individual or individuals who happen to be doing the hiring’ [121].

The early 1940s saw the reemergence of personality testing. One of the most well-known test devised in this period was created by Katherine Cook Briggs and Isabel Briggs Myers [122], who based their theories on the Jungian archetypes [123]. This mother-and-daughter pair was mainly motivated by the need for an instrument to aid executives and managers to evaluate female candidates for various factory and industrial positions during World War II. They devised a type indicator, known as the Myers-Briggs Type Indicator, which was a pen-and-pencil test and once scored was used as a means to allocate the most suitable employee to a particular task but also as a method for conflicts resolution. After World War II, the use of personality testing shifted its focus on assessing employees most suited for managerial and executive positions and many companies, such as IBM, started developing their own employee personality tests. However, during the 1960s, testing restarted being practised at all levels and over a wider variety of occupations. Today, as much as 60 per cent of companies adopt employee personality testing – albeit for different reasons and for gathering different information. There are also many instruments available to help employers consider factors other than personality, such as cognitive ability, integrity, job knowledge and physical ability.

8.2.1 Overview of personality tests

Personality testing has come a long way since Woodworth’s Personal Data Sheet. The intensive research in the field has led to many additions and modifications, both in approach and application. The most widely administered test is the Myers-Briggs Type Indicator (MBTI) [122] that scores individual preferences based on the works of Carl Gustav Jung [123]. Individuals answer a psychometric questionnaire that assesses preferences relating to four dichotomies:

- Extraversion/Introversion (E/I), which concerns the attitudes or orientations of energy. This dichotomy assesses whether the energy and attention of an individual is directed either towards people and objects in their environment or towards ones internal world, experiences and reflection.

- Sensing/Intuition (S/N), which relates to the functions or processes of perception. This dichotomy evaluates whether an individual carries out information gathering either by observing using the five sensory attributes or by perception of future possibilities and meanings.

- Thinking/Feeling (T/F), which refers to the functions or processes of judging. This dichotomy determines whether an individual performs decision making by either linking ideas together through logical connections or by weighing relative values and issues.

- Judging/Perceiving (J/P), which is associated with the attitudes or orientations towards dealing with the outside world. This dichotomy establishes whether an individual is either concerned with making decisions or is attuned to incoming information.

The personality type of an individual is determined by which alternative of each dichotomy is preferred. It should be noted that preference of one option does not mean that the other is never used – it is simply regarded as being less preferred. As a result there are 16 possible combinations of personality types as shown in Table 8.1

Table 8.1: The 16 personality types of the Myers-Briggs Type Indicator

ISTJ *Dependability*	ISFJ *Commitment*	INFJ *Integrity*	INTJ *Vision*
ISTP *Ingenuity*	ISFP *Sensitivity*	INFP *Idealism*	INTP *Logic*
ESTP *Energy*	ESFP *Enthusiasm*	ENFP *Imagination*	ENTP *Initiative*
ESTJ *Decisiveness*	ESFJ *Affiliation*	ENFJ *Responsiveness*	ENTJ *Drive*

Another well known personality test is the Keirsey Temperament Sorter [124], which is closely related to the MBTI, but instead uses temperaments rather than attitudes and functions. The four temperaments (artisan, guardian, idealist and rational) can be subsequently broken down into roles and further into role variants as shown in Table 8.2.

The temperaments are separated firstly by whether a person prefers to perceive (observant) or reflect on their inner world (introspective). An individual can then be described by what they pay attention to the most. If a person is more concerned about other peoples' feelings then they are cooperative, whereas those who pay attention to their own thoughts and emotions are said to be pragmatic. Communicating with others distinguishes whether an individual prefers to inform others (informative) or direct others (directive). Each temperament makes this distinction and results in the eight roles. Finally, the way with which a person relates to their environment leads to the 16 role variants. A person who acts before reflecting is considered as expressive, whereas a

Table 8.2: The breakdown of the Keirsey's temperaments into roles and role variants

	Temperament	**Role**	**Role variant**
Abstract or Concrete?	**Cooperative or Utilitarian?**	**Directive or Informative?**	**Expressive or Attentive?**
Observant	Artisan	Entertainer	Composer Performer
		Operator	Crafter Promoter
	Guardian	Administrator	Supervisor Inspector
		Conservator	Protector Promoter
Introspective	Idealist	Advocate	Champion Healer
		Mentor	Counsellor Teacher
	Rational	Coordinator	Field marshal Mastermind
		Engineer	Architect Inventor

person who reflects before acting is thought to be attentive. An observant, cooperative, directive and attentive person would be classed as having a supervisor temperament.

Many similarities can be seen between the KTS and the MBTI (shown in Table 8.3). However, there are substantial differences in both theoretical and practical aspects, as well as the administration of the two tests.

Table 8.3: KTS role variants with corresponding MBTI types

Inspector ISTJ	Protector ISFJ	Consellor INFJ	Mastermind INTJ
Crafter ISTP	Composer ISFP	Healer INFP	Architect INTP
Promoter ESTP	Performer ESFP	Champion ENFP	Inventor ENTP
Supervisor ESTJ	Provider ESFJ	Teacher ENFJ	Field marshal ENTJ

Finally, the NEO Inventories [125] are a collection of psychological personality assessment instruments that measure the Five-Factor Model (FFM) personality traits of individuals [126], also known as the 'Big-Five' personality traits. Specifically, these instruments are used to determine

emotional, interpersonal, experiential, attitudinal and motivational styles represented by five domains of personality and their facets:

- Neuroticism, which reflects the level to which an individual is predisposed to experiencing negative emotions, such as sadness, embarrassment, fear and anger.
- Extraversion, which refers to the level to which an individual engages with their external world through interpersonal interactions, as well as their energy and predisposition to experiencing positive emotions.
- Openness to experience, which concerns an individual's tendencies regarding intellectual curiosity, creativity and variety in interests and experiences.
- Agreeableness, which involves interpersonal orientation with regards issues, such as compassion, social harmony, cooperation, and trust.
- Conscientiousness, which relates to the degree of self-discipline and control of impulses, and also ambition and organization.

The facets defining each domain can be seen in Table 8.4. Currently, there are several inventories that can be administered. The NEO-PI-R/NEO-PI-3 inventories are a 240-item questionnaire that contains statements describing various behaviours. An individual answers the questionnaire using a five-point Likert scale by simply indicating the degree to which he or she agrees or disagrees with the statement given in each question. For each facet, there are eight related questions that need to be answered, giving a total of 48 questions per domain. Based on the responses of the individual, a profile of his or her personality is formed containing the scores measured for each facet and domain separately. A shorter 60-item questionnaire can also be administered using the NEO-FFI-3, which only provides scores of the five personality domains (each measured on a 12-item scale), but not on the facets defining each domain. The purpose of this inventory is to give a brief and comprehensive assessment of the personality of an individual.

8.2.2 Advantages and disadvantages of employee personality testing

According to the Society of Industrial and Organizational Psychology Inc, Division 14 of the American Psychology Association [127], employee testing can yield a number of benefits. First of

Table 8.4: The personality domains and facets of the NEO Inventories

Neuroticism (N)	**Extraversion (E)**	**Openness to Experience (O)**
Anxiety (N1)	Warmth (E1)	Fantasy (O1)
Angry hostility (N2)	Gregariousness (E2)	Aesthetics (O2)
Depression (N3	Assertiveness (E3)	Feelings (O3)
Self-consciousness (N4)	Activity (E4)	Actions (O4)
Impulsiveness (N5)	Excitement-seeking (E5)	Ideas (O5)
Vulnerability (N6)	Positive emotions (E6)	Values (O6)

Agreeableness (A)	**Conscientiousness (C)**
Trust (A1)	Competence (C1)
Straightforwardness (A2)	Order (C2)
Altruism (A3)	Dutifulness (C3)
Compliance (A4)	Achievement-striving (C4)
Modestry (A5)	Self-discipline (C5)
Tender-mindedness (A6)	Deliberation (C6)

all, testing saves time during the hiring process as it can reduce the time spent interviewing candidates by excluding applicants that are less skilled, or lack abilities or characteristics of what the job requires. It is, therefore, considered a fair and cost-efficient method of hiring within a reasonable amount of time especially with a very large number of applicants. Another advantage to employee testing is that it limits erroneous hiring decisions by management executives, which can be very expensive for a company when taking into account possible training expenses, low performance costs and the costs of replacement. Additionally, most tests involve measuring characteristics, such as personality traits and extensive know-how, which are gained through experience rather than some sort of training. Also, employee testing can be more successful and less expensive than otherwise traditional methods when trying to attain certain difficult information on candidates. Some may consider certain hiring processes to be subjective if the information gathered on applicants is not the same or if the information is not used in the same way. Employee testing, on the other hand, introduces a standardized approach ensuring that this bias is removed or at least kept to a minimum, and that all applicants are handled in the same manner.

One of the disadvantages of employee testing is that it cannot always be administered. For instance, when the instrument used for testing needs to be customized, then the administering costs can add up. However, the future benefits for a company using testing for hiring applicants or retaining employees may prove to be a sound investment. Another significant drawback to employee

testing is the possibility of an applicant or employee taking legal action for discrimination. In particular, differences in demographic groups may be highlighted more in certain test scores, and so selection based on these tests could be deemed exclusionary by some participants. In addition, personality testing can only be carried out if the proper resources are available. This includes the right people, as well as a suitable period of time to carry out the test. Finally, it is not suitable when the number of candidates is relatively low.

Investigating whether or not a specific type of job can be executed by a particular personality type, especially for the heavily people-oriented field of software development, is very appealing to many organizations. It can help supervisors decide on issues such as pay rises and promotions or, in a negative light, disciplinary action and dismissal. As a result, several studies have been carried out to investigate whether software development professionals possess a specific type of personality. The outcomes of these investigations can shed light on the type(s) of personality that are drawn towards a career in software development. It also enables to explore whether different professions within the industry appeal to different personality types.

8.3 Personality types of software development professionals

One of the earliest studies of personnel in software engineering-related occupations was performed by Moore [128]. The study was based on the 16 Personality Factor Questionnaire (16PF) [129], a popular measurement tool in personality-occupation studies, which is extensively used to assemble personality profiles for people in various occupations [130]. In the study, the author compiled the 16PF questionnaire for four software development occupation categories (namely, application programmers, systems analysts, technical programmers, and data processing managers) in an attempt to answer the question: '*Do these groups of information systems professionals share a common personality profile, or are there significant differences?*' After multiple analyses, the authors found that managers and application programmers were most similar in that they are more inclined to experiment and think freely, thus allowing them to use their imagination more, while at the same time being more outspoken and comfortable with whatever happens. In contrast with application programmers, however, managers are more likely to be laid-back and spontaneous, more forceful and competitive, and more capable of abstract thinking. Another finding showed that systems analysts and technical programmers have a tendency to be more practical, careful and conservative than data processing managers because their work is often highly visible, not only

within data processing but throughout the company. Mistakes can be costly, but also embarrassing for them. Additionally, the study also identified managers as '*less concerned with social rules than most people*' and more likely to pursue their own desires.

Wynekoop and Walz attempted to explore the differences between information systems professionals in order to determine whether or not differences existed in personality characteristics with the rest of the general population [131]. They surveyed three oil and gas companies with a total of 114 programmer analysts, systems analysts, and project managers by administering the California Psychological Inventory Adjective Check List (ACL) [132] on the employees. The results showed that managers and system analysts are more similar to each other than to programmers. In addition, managers and system analysts differ from the general population on more scales than programmers but also on different scales. Another finding was that managers tend to be more logical, compliant and with more confidence than the general population, whereas analysts are more willing to keep friendly relationships with others. Generally, the study shows that IT professionals have more leadership skills, are more ambitious and reasonable and have more self-esteem and can be more disciplined than other professionals. Similarly, Smith [133] also carried out research on IT professionals, though his work concentrated only on the personality types of systems analysts. Based on MBTI type tests, the author concluded that a high majority of systems analysts tend to prefer sensing and thinking, in addition to being more introvert rather than extrovert.

Capretz attempts to provide a personality profile of software engineering employees by distributing the MBTI instrument to 100 software engineers working for the government, for private companies and students of private or public universities and comparing results to the distribution of MBTI types of the general US adult population [134]. The motivation behind the author's research is the fact that the majority of software engineering professionals are typecast as a '*nerd*' – an introvert working alone in his corner and with no intentions to interact with others. However, over the years software development has become more complex leading to a rise in the need for specialization within the profession, such as system analysts, designers, programmers, testers, etc. As a result, the author believes each role requires a corresponding personality type. Furthermore, at the time of the study, there had been very little research carried out regarding the degree of job satisfaction among software professionals and any profile developed of the software engineer may have been modified due to the growth of the field's popularity. The results of the author's survey showed that, of the 16 possible MBTI types, the most dominant were ISTJs, ISTPs, ESTPs, and

ESTJs accounting for around 55% of the total sample. The majority of software engineers fall under the ISTJ personality type (24%), which is a higher percentage than that of the general population of the US (11.6%). This is largely due to the fact that the majority of software engineers are technically oriented and prefer working with the facts and reason rather than with people. It was also noted that systems analysts possessed an extroverted thinking personality type, which prefers to communicate with other people and use their enhanced thinking ability in order to solve organizational problems. On the other hand, programmers exhibit an ISTP personality type, which excels at pinpointing the root of a problem, as well as coming up with practical solutions (regarded as '*gurus*'). Conversely, '*wizards*' possess an INTJ personality type, exhibiting a great desire to achieve things but unwilling to socialize with others. Introverts make up the majority of software development professionals; as such they usually find it hard to communicate with end users. The greatest difference, according to the author, between software engineers and the general population is that the majority of software engineers '*take action based on what they think rather than what somebody else feels*'. This, however, does not help reduce the communication gap between software developers and end users. A more recent comprehensive investigation can be found in an analysis by Varona, et al., which surveys existing studies that try to profile software development professions, in order to properly understand the human resources working in the software industry, as well as to spot possible trends and changes [3].

8.3.1 Allocating developers to tasks based on personality type

Even if a specific personality type can be distinguished for each software development profession, the most important question is how to make use of this information in practice when trying to allocate and schedule human resources or form software development teams. There has been a gradual rise in the number of approaches aiming to help answer this question, which are presented here.

The personality type of a developer can play a significant role in determining which tasks he or she is assigned to because particular individual traits can help certain developers to be more adept in coping with the requirements and characteristics of a specific task. Furthermore, a more suitable personality type assigned to a task can have a direct influence on individual performance and team efficacy [135, 136], as well as group conflict and team cohesion [137]. It also can contribute to the overall quality of the final software product [138]. In addition, when a developer is assigned to a

task that suits his or her personality, then his or her level of job satisfaction can increase leading to higher productivity [139].

Dafoulas and Macaulay's approach to assigning developers to tasks uses dynamic role allocation to maximize productivity and performance and takes into account certain role criteria (such as the goals and objectives, skills and knowledge, as well as any personality and culture requirements) so that project managers can assign/reassign roles or activities to team members according to their suitability [140].

Acuña and Juristo also consider roles and human capabilities in their attempts [141]. Their proposed model first determines the intrapersonal, organizational, interpersonal and management capabilities of team members and then performs role assignment to team members based on the capabilities required by the roles and the capabilities of the available resources. Each capability is allotted a number of personality traits required to be possessed using the 16PF test as a psychometric instrument. The goal is to assign those employees possessing the personality traits nearest to those required by the role [142].

Similarly, André et al.developed a formal model for human resource allocation focusing on the assignment of developers to roles [143]. In this approach, rules are generated to undertake the team formation process based on the roles and competencies of developers assessed through psychological tests. These team formation rules were converted into a formal model comprising four objective functions (competence, team compatibilities, availability and distance cost) and twelve constraint types, to perform human resources assignment to roles by employing heuristic algorithms (random restart hill-climbing, simulated annealing, Tabu search, and various other combinations of heuristic approaches).

Capretz and Ahmed presented an attempt at human resource allocation suggesting a mapping of job requirements and skills to personality characteristics of employees, stating that diversity of psychological types improves effectiveness and fulfilment of software developers [144, 136]. Because employees are more likely to perform better if they are assigned roles that their personality traits are best suited to, the authors associate hard skills (in the form of job requirements) to soft skills (in the form of personality requirements) for various software professionals: systems analysts, designers, programmers, testers and maintenance staff. The soft skills are then matched with specific personality characteristics based on MBTI personality types and this can allow project

managers to select team members with the same personality types and assign them to the roles required in the project.

8.3.2 Allocating developers to teams based on personality type

While some researchers argue the importance of getting a developer to work on the right task, others argue the significance of getting developers to work right together. The approaches mentioned in Section 8.3.1 all focus on the relationship between developers and tasks. However, other approaches focus on the relationships among developers in order to try to identify how the personality types of developers influence various facets of team work and investigate whether certain combinations of personality types improve aspects such as performance, productivity and even quality. From a software project manager's perspective, this could help him or her to understand and exploit this underlying factor effectively when deciding on assigning developers to tasks.

One area of study concerns the heterogeneity of personality types, that is, the diversity of traits possessed by developers. Rutherfoord examined the impact of diversity by comparing teams comprising different personality types with teams composed of the same personality type using the KTS [145]. The results showed that groups with members of the same personality type were having more personal problems, rather than technical. The surveys revealed that members seemed to want to elaborate the project by themselves and had problems with members that did not have much of a sharing discipline. On the other hand, groups with members of different personality types seemed to have more problems at a technical level. It was also noticed that groups where all members possessed a supervisor personality type were spending too much time discussing how tasks will be assigned, despite this matter having already been decided previously. Groups where all members possessed an inspector personality type were very quiet and interaction between them did not seem to exist. These groups appeared, however, much more focused and responsible. Groups with different personality types among their members were very active, had robust discussions and provided different kinds of ideas. The authors also noticed that groups with supervisor personality types were very opinionated and preferred to 'follow a traditional path'.

Research by Neuman et al. investigated the relationship between work team effectiveness and two other factors: team personality elevation, defined as 'the average level of a given trait within a team', and team personality diversity, described as 'the variability or differences in personality traits found within a team' [4]. Predicting job performance using personality has conventionally

been based only on the elevation, or magnitude, of traits within a group and this has been the foundation of selection and placement strategies. Nevertheless, the authors claim that team-based designs may also require taking into account the diversity, or variability, of traits within a group, in order to find correlations between personality and job performance. The research used the FFM to examine the relationship between team personality composition and work-team performance. Based on the authors' interpretation of the results, teams perform better when members differ in terms of extraversion and emotional stability (neuroticism) rather than when members are similar in terms of these traits. Conversely, team performance is likely to increase if team members possess similarly high levels of traits regarding conscientiousness, agreeableness and openness to experience. Therefore, project management decisions on developer selection can be supported by taking into account the similarity of certain traits and the dissimilarity of others within a team.

A similar study was conducted by Pieterse et al., where the authors developed a methodology to investigate the role of personality diversity in teams [146]. Specifically, they came to the result that personality diversity has more positive influence on team performance compared than team's capabilities. They also support that personality diversity yields faster team productivity. Nonetheless, they caution that diversity alone should not be taken as a sure predictor of a team's performance, and other factors should also be taken into consideration, such as the ability of leadership, team communication and group cohesion. Peslak reports that personality is significantly related to team success because it improves team cohesion, communication to handle conflicts, creates a more pleasant atmosphere and team roles [147]. All of these impacts were found to correlate to increased team performance. Whether or not personality diversity helps team performance, the paper supports the opinion that 'perhaps the advantages of diversity are offset by the conflicts that can arise'.

A large number of research studies concentrate on the effects of personality in agile methodologies, which is a relatively new development approach in the field of software engineering. Agile methodologies transform the way in which communication, collaboration and coordination practices in software development projects are carried out towards a more people-oriented approach, where software teams are self-managing and share the decision making. Approaches here focus particularly on pair programming activities and how personality types are implicated. These activities involve two developers working together on one task as they alternate between the role of 'driver' – the developer who codes – and 'navigator' – the developer who reviews the code.

Immediately, there is a need for social interaction (in the form of communication, collaboration and cooperation) among the developers in order to reach a common goal of delivering the unit produced on time and with the required quality. Hence, this is the reason why, especially in the last five years, investigations have been carried out to investigate the impact that personality types have on the performance and productivity of the pairs. For example, Sfetsos et al. investigate the diversity of the personality traits of pair programmers and came to the conclusion that pairs with heterogeneous personalities and temperaments exhibit better performance and collaboration-viability than pairs with similar personality traits [148]. Software project managers, therefore, can take into account personality types when allocating developers to tasks and try to match developers so as to optimize the pair's effectiveness. Similarly, Choi et al. investigate which combination of personality types yields higher pair productivity [149]. Specifically, they tested pairs of developers with alike, opposite and diverse combinations of personality types and found that the latter combination outperformed, in terms of code productivity, the other two.

An other attempt investigating the diversity of personalities can be also be found in the study by Karn et al. [150]. The MBTI instrument was administered to determine the personality types of developers, in addition to various other questionnaires to identify the level of cohesion of software development teams following the Extreme Programming (XP) methodology. Workgroup cohesion is defined by the authors as 'the degree to which team members have close friendships with others in their immediate work unit and their personal attraction to members of the group'. The authors support the argument that certain combinations of personality types work better and decided to investigate development teams of university students working on software projects for external clients selected on personality type, nationality and previous skills/experience. Some interesting results were observed. First, the team with the highest level of cohesion did not have the highest performance rating. This finding means that cohesion is not the only factor contributing towards high performance and subsequently team success, but increases the possibility of a team to become a successful. Another observation was that the team with worst level of cohesion could not be characterised by a typical personality team type. Also, teams having a more traditional science/engineering personality profile seem to outperform those with more diverse types. Overall the authors summarized that a combination of personality types is important and that they factor in both team performance and cohesion.

In practice, a software development company may find it easier and cheaper if developers are left to team up by themselves. Oftentimes, however, pairs will be formed based on friendships and common interests, and not on optimizing productivity. If personality types are taken into account, a software project manager can assign tasks to developers yielding maximum effect with relatively little time and cost.

Acuña et al. explore the relationship between each of the five factors of the FFM and job satisfaction, performance, team cohesion, task conflict and quality in agile settings [139]. Results of their quasi-experiment produced a variety of results. Firstly, they observed that the quality of the end product is positively correlated to the preferred interpersonal style of the developers. This means that teams with a high average level of extraversion will enjoy the social interaction that is promoted through agile methodologies and all members share the same goal of making the project a success. They also noted that developers with positive attitudinal and motivational styles are also more likely to be satisfied with their job. Developers in a team that share the same high level of agreeableness and conscientiousness feel more content with their career. Staying with the factors of the FFM, Salleh et al. explored how they especially affected pair programming [151]. The main findings here were that pairings of developers with high levels of traits relating to openness to experience were conducive to the effectiveness of the pairings. Hannay et al. (2010) provide a comprehensive survey of the research investigating the effects of personality on pair programming and its ability to predict job performance.

8.3.3 Discussion

There are two schools of thought concerning the inclusion of information regarding the personality types of developers for human resource allocation and scheduling activities in software development. On the one hand, there is a view that a developer should be assigned to a task that he or she is more suitable based on the requirements of the task and the personality type of the developer. The claim is that each software development task has a set of characteristics and requirements that can be associated to a desired set of personality traits. For example, requirements elicitation tasks involve a high level of social engagement and the ability to identify with clients to understand their needs. Therefore, an introverted individual may struggle to perform these tasks, as they are more reserved and prefer working alone rather than in environments requiring high social interaction. The research does not claim that a developer cannot be able to carry out a task

if he or she does not have the right personality type; it claims that he or she would not prefer to carry out the task. Consequently, a better task-fit for a developer would result in, not only better performance, but also higher job satisfaction. The more fulfilled a developer is while working on a task that he or she is suited to, the more productive and efficient he or she is. Of course, this can only work if the developer is capable of carrying out the task in the first place with regards technical skills, knowledge and expertise, so as not to jeopardize the quality of the software being developed.

On the other hand, there is the standpoint that a developer should be assigned with other developers so that the resulting combination of personality types leads to increased performance and effectiveness. The claim here is that there are certain combinations of personality traits that can improve the productivity of the team and increase probability of success. Some traits, such as conscientiousness, should be present in all team members, while other traits, such as extraversion, should be diverse. Contrariwise, if several developers are assigned to work together on a task, their combination of personality types may not foster the most efficient and productive working environment. This does not mean that the job cannot get done; it may just mean that a more appropriate mixture of developers in terms of personality type may be able get the job done with improved levels of communication, collaboration and coordination, which are governed by an individual's personality type. Inevitably, if this personality type blend is not 'effectual', there will be several knock-on effects, such as lowered productivity, job satisfaction, and, ultimately software quality.

Overall, there are a limited number of approaches that attempt to incorporate personality types of software developers in order to perform resource allocation, which is expected as this is still a relatively new direction. Some approaches, for example, do not treat resource allocation and task scheduling as an operational research problem and therefore do not employ the specialized techniques and methods as the ones presented in Chapter 3. Those that do, attempt to only optimize the allocation of resources so that tasks are assigned developers whose personality type is closest to a desired profile. Interestingly, all but one approach overlook dealing with the problem of task scheduling altogether, which as previously mentioned is tough to separate from resource allocation since both activities are affected by developer assignment constraints. Hence, the ability of approaches to provide an integrated tool may be considered limited unless they are able to accommodate scheduling (or scheduling constraints) also.

8.4 Further research trends and challenges

Incorporating aspects of personality types in allocation and scheduling is still at a young and exploratory stage, and so the applicability of approaches lacks the back-up of empirical evidence demonstrating their practical benefits in order to promote their adoption by real-world software development companies. A systematic evaluation of the effect is still required to be carried out to gather such evidences and, if these continue to indicate promising results, only then can a significant evolution in team formation, as well as allocation and scheduling strategies, occur.

The desired personality types of roles, tasks or activities that form the basis of assigning suitable developers in a number of approaches are not always justified empirically. It is important that desired personality type of a task is correctly identified in order to allocate a suitable developer, but this may pose a challenge given the different personality measures and frameworks available to assess personality types and preferences. There is currently no consensus as to which personality instrument is the most capable of providing a task's desired personality accurately.

Considering the use of personality types does not aim to single out developers or discriminate against them. Instead, it is supposed to provide software project managers with additional and complementary information to help them in the allocation and scheduling of resources or, in general, task-independent team formation. Additionally, it should not substitute or force to disregard important technical factors such as knowledge, skills and experience. However, some developers may still consider such approach intrusive, so it is therefore important to provide reassurances that the goal is to utilize this human-centric factor to achieve maximum resource usage through the strengths of developers. Ultimately, the goals and objectives of any approach is to eliminate those risks in software project management preventing development organizations from delivering their products on time, within budget and with the required level of quality.

There is a differing of opinions with respect to how personality types can be utilized – for assigning tasks or for staffing teams. Either way, the emphasis remains on gathering evidence whether taking into account personality types of developers constitutes a legitimate way forward to help software project managers make staffing decisions aiming to increase the probability of success. Ideally, future approaches will be able to support both these valid research viewpoints.

Chapter 9 describes an approach developed as part of the research work, which takes into account the personality type of software development professionals for human resource management. More specific, an optimization approach is presented that attempts to allocate developers to

tasks in a way that tasks will be assigned to developers who are better suited with respect to their personality types but also takes into account the levels of experience of the available resources.

Chapter 9

Approach 5: Optimizing Developer Experience, Developer Personality Type and Team Size using a Pareto Ranking Genetic Algortihm

The goal of this proposed approach [152, 153] is to use a Pareto ranking genetic algorithm to optimally allocate resources to tasks so that the tasks are carried out by: (1) the most experienced developers based on technical knowledge and skills, (2) the most suitable developers based on personality traits and abilities, and (3) the smallest number of developers possible so as to avoid over-assignment. These criteria are expressed in the form of objective functions, which take into account the skills and personality type required by tasks, in addition to the skills and personality type possessed by developers. Furthermore, the approach assumes that developers assigned to tasks must have the required skills to carry out the tasks, and also that developers must only be assigned to one task at any given time so as to avoid assignment conflicts. The objective functions along with the constraint functions form the basis of evaluation of solutions. In order to handle the possibly competing nature of the objectives, the Non-dominated Sorting Genetic Algorithm (NSGA-II) is again adopted as the optimization method [89]. Unlike the previous approaches proposed in Chapters 4–7, the duration of the project is not taken into account as an objective to help schedule tasks. Instead, the focus of the approach is solely on allocating resources to tasks that have already been scheduled.

Personality type in this proposed approach is based on the personality domains of the Five-Factor Model (FFM) described in Section 8.2.1 [126]. The FFM has been widely adopted in many academic and application disciplines where personality measures have been required, and is a common instrument in cases involving career and personnel assessment. Specifically, to measure the

personality type of a developer, the 60-item NEO-FFI-3 inventory is administered [125], which scores each developer on the five domains: neuroticism (N), extraversion (E), openness to experience (O), agreeableness (A) and conscientiousness (C).

9.1 Research questions

The main research target is to investigate the quality of the resource allocation solutions that are generated with respect to the objectives and constraints involved in the optimization. In an ideal case, objectives will not be competing at all: developers possessing a high level of experience in a certain skill of a profession will also possess the most suitable personality traits of that particular profession. On the other hand, in the worst case, objectives will be highly competing: developers possessing a high level of experience in a certain skill of a profession will possess the least suitable personality traits required for that particular profession, and vice-versa. Therefore, it is important to evaluate how well the objective functions guide the algorithm to generate (near-)optimal solutions in both these cases.

Specifically, the first research question (RQ5.1) assesses how well the objective functions perform in generating (near-)optimal solutions in a best-case scenario where highly experienced developers also possess the most suitable personality type: **RQ5.1 How well does the algorithm perform in generating (near-)optimal solutions in cases where objectives are not at at all competing?**

Conversely, the second research question (RQ5.2) assess how well the objective functions perform in generating (near-)optimal solutions in a worst-case scenario where highly experienced developers posses the least suitable personality type, and vice-versa: **RQ5.2 How well does the algorithm perform in generating (near-)optimal solutions in cases where objectives are highly competing?**

Several experiments using different-sized software project instances are used to help answer these research questions.

9.2 Problem description

A software project consists of a set $T = \{t_1, t_2, \ldots, t_m\}$ of m tasks. Each task t_i is associated with one and only one profession from the set $H = \{h_1, h_2, \ldots, h_q\}$. In order to be completed, each task requires a number of skills, which are related to the profession with which the task is

associated. The complete set of S skills required by a project is given by $S = \{s_1, s_2, \ldots, s_p\}$. As previously mentioned, the approach assumes that all tasks have already been scheduled. Hence, each task has an associated start time, t_i^{start}, and finish time, t_i^{finish}.

A development company has n developers available to carry out a project, which are given by the set of $R = \{r_1, r_2, \ldots, r_n\}$. The level of experience each developer possesses in all skills is represented using the matrix $LEXP = [lexp_{jk}]$, where $lexp_{jk} \in [0, 1]$ and denotes the level of experience that developer r_j possesses in skill s_k. The lower the value of $lexp_{jk}$, the lower the level of experience possessed by the developer in the skill, and vice-versa.

To be able to compare the personality type of a developer to that desired by a profession, each personality domain of the FFM is associated with a level (1:low, 2:average or 3:high) corresponding to the degree to which each domain is either possessed by a developer or desired by a profession.

Specifically, the personality type of developer r_j is denoted by a vector r_j^{pers} consisting of five elements – one for each personality domain – such that $r_j^{\text{pers}} = \{r_j^{\text{N}}, r_j^{\text{E}}, r_j^{\text{O}}, r_j^{\text{A}}, r_j^{\text{C}}\}$, where each element represents the level to which the corresponding domain is possessed: r_j^{N} for the level of neuroticism, r_j^{E} for the level of extraversion, r_j^{O} for the level of openness to experience, r_j^{A} for the level of agreeableness and r_j^{C} for the level of conscientiousness. For example, if the personality scores of developer r_j indicate low neuroticism, high extraversion, average openness to experience, high agreeableness and low conscientiousness, then the personality type will be expressed as $r_j^{\text{pers}} = \{1, 3, 2, 3, 1\}$.

Similarly, the desired personality required by profession h_l is also denoted by a vector $h_l^{\text{pers}} = \{h_l^{\text{N}}, h_l^{\text{E}}, h_l^{\text{O}}, h_l^{\text{A}}, h_l^{\text{C}}\}$, where each element in the vector reflects the desired level of the corresponding domain that is required by the profession.

An important part of the research carried out in this approach involved deciding which personality type is desired for specific software development professions. The professions investigated were previously presented in Table 7.2, which were determined from the Standard Occupational Classification (SOC) System [111]. A detailed analysis of these nine professions was carried out using the O*NET Resource Center [112], which provides a content model and an on-line database defining standardized and occupation-specific descriptors of each profession using the SOC System coding. Each occupation's job-related and worker-related characteristics and requirements

were retrieved containing information on: the abilities and work styles of workers, the skills required by workers, and the work activities of occupations. Once these key requirements and characteristics were identified, the most suitable personality traits required by developers to carry out activities of each profession were then associated with corresponding personality traits of the FFM. The level of each domain required by the professions are presented in Table 9.1/

Table 9.1: Desired level of each FFM personality domain for the nine identified software development-related professions

		Desired personality domain level				
SOC code	**Profession**	**N**	**E**	**O**	**A**	**C**
11-3021.00	Computer and information systems managers	Low	High	Average	Low	High
15-1121.00	Computer systems analysts	Low	High	Average	High	Average
15-1122.00	Information security analysts	Low	Low	High	Low	High
15-1131.00	Computer programmers	Low	Low	Average	Low	Average
15-1134.00	Web developers	Low	Average	High	Average	Low
15-1143.00	Computer network architects	Low	Low	High	Low	Low
15-1199.01	Software quality assurance engineers and testers	Average	Average	Low	Low	Low
15-1199.02	Computer systems engineers/architects	Low	High	High	Low	Average
15-1199.06	Database architects	Low	Low	High	Low	Low

9.3 Pareto ranking genetic algorithm method

The basic steps of the NSGA-II algorithm is given in Section A.3. In brief, the algorithm attempts to find a set of (near-)optimal solutions by using a nondominating sorting procedure and a crowding comparison operator. The former helps to promote individuals whose fitness values are ranked higher, whereas the latter helps promote individuals that can potentially improve the diversity of the population despite having a lower rank.

9.3.1 Representation and encoding

This approach only attempts to allocate resources to tasks based on the characteristics of the required tasks and available developers; therefore the objective functions require individuals to contain information regarding which developers are assigned to each task. Specifically, each individual, x, in the population is represented by a bitset array of length m. Each element i of the array contains a bitset, $B^i = \{b_1^i, b_2^i, \ldots, b_n^i\}$, of length n that represents the developers assigned to carry out task t_i. Each bit in a bitset represents one specific developer such that if bit $b_j^i = 1$ then this indicates that developer r_j is assigned to work on task t_i. Else, if bit $b_j^i = 0$, then this

denotes that developer r_j is not assigned to work on task t_i. Overall, each individual requires a total of $m \times n$ bits. Figure 9.1 gives an example of how solutions are represented by individuals in a population. Individuals are encoded using a bitset array with five elements – one for each task, and each element contains a bitset consisting of six bits – one for each available developer. Therefore, a total of 30 bits are required to represent solutions.

$T01$	$T02$	$T03$	$T04$	$T05$
001101	101001	100100	000101	111000

Figure 9.1: Example of the representation and encoding of individuals used in Approach 5

9.3.2 Population initialization

The population is generated in a way that ensures all individuals are feasible by having at least one developer assigned to each task. For each element of an individual's array, a randomized bitset is created. If all bits have a value of 0, then another bitset is randomly generated. This process is repeated until at least one of the bits has a value of 1.

9.3.3 Optimization function

A total of three objective functions were created for the evaluate the fitness of each individual, along with two constraints to measure the degree of feasibility of each solution. The optimization function is given in Equation (9.1).

$$\begin{aligned} \underset{\forall x \in POP}{\text{Maximize}} \quad & \mathcal{F}(x) = (\mathcal{F}_{\text{experience}}(x), \mathcal{F}_{\text{personality}}(x), \mathcal{F}_{\text{size}}(x)), \\ \text{subject to} \quad & \mathcal{G}_{\text{skills}}(x) = 0, \\ & \mathcal{G}_{\text{assignment}}(x) = 0. \end{aligned} \tag{9.1}$$

9.3.3.1 Objective functions

9.3.3.1.1 Developer experience objective function

The first objective function is responsible for ensuring that highly-experienced developers are assigned to each task. The approach adopts the function described in Equation (4.9) in order to evaluate the fitness value of each individual x in the population concerning this objective.

$$\mathcal{F}_{\text{experience}}(x) = \frac{1}{m}\sum_{i=1}^{m} t_i^{\text{experience}}. \qquad \text{(4.9 revisited)}$$

9.3.3.1.2 Developer personality objective function

The second objective function is used to allocate the most suitable resources to each task based on the personality type required by the profession that each task belongs to. To calculate the fitness value of each individual x in the population, the personality type of each developer is matched against the desired personality type of the profession. This matching is carried out by calculating the distance between the level at which each domain is possessed by an assigned developer r_j and the level at which the domain is desired for the profession h_l that task t_i belongs to. This personality distance is given by

$$r_j^{\text{dist}} = |r_j^{\text{N}} - h_l^{\text{N}}| + |r_j^{\text{E}} - h_l^{\text{E}}| + |r_j^{\text{O}} - h_l^{\text{O}}| + |r_j^{\text{A}} - h_l^{\text{A}}| + |r_j^{\text{C}} - h_l^{\text{C}}|. \qquad (9.2)$$

Then, by using Equation (9.3) the average personality distance of all developers assigned to task t_i is computed.

$$t_i^{\text{dist}} = \frac{1}{t_i^{\text{assigned}}} \sum_{\forall r_j \in A^i} r_j^{\text{dist}}. \qquad (9.3)$$

The objective function of Equation (9.4) proceeds to average the values of the personality distance over all tasks in the project. The lower the partial values of t_i^{dist}, then the more fit the individual with regards this objective due to the shorter distance between the desired personality type of the task and the personality type possessed by the assigned developers.

$$\mathcal{F}_{\text{personality}}(x) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{1 + t_i^{\text{dist}}}. \qquad (9.4)$$

9.3.3.1.3 Team size objective function

The third objective function is employed to handle the over-allocation of resources to tasks by attempting to minimize the number of developers assigned to each task of the project. It makes use of the function defined by Equation (6.3) in order to compute the fitness value of each individual x in the population with regards to this objective.

$$\mathcal{F}_{\text{size}}(x) = \frac{1}{m}\sum_{i=1}^{m} t_i^{\text{size}}. \qquad \text{(6.3 revisited)}$$

9.3.3.2 Constraint functions

The assumptions made in this proposed approach that govern the feasibility of solutions concern: (1) the satisfaction of skills, and (2) the availability of developers. For the former, the required skill constraint function is employed to evaluate the extent to which skills required by tasks have been satisfied by the developers assigned. For the latter, the developer availability constraint function is used to evaluated the degree to which developers have been conflictingly assigned to more than one task at any given time in the project.

9.3.3.2.1 Skill fulfilment constraint function

In regards to the fulfilment of skills, a solution is considered feasible only if each skill required by a task can be satisfied by at least one of the developers assigned to the task. For each individual x in the population, the evaluation is carried out using the function previously defined in Equation (5.7), which aggregates the ratio of unsatisfied skills to total number of required skills calculated for each task t_i, to give the corresponding solution's degree of feasibility with respect to this constraint.

$$\mathcal{G}_{\text{skills}}(x) = \frac{1}{m}\sum_{i=1}^{m} t_i^{\text{sviolations}}. \qquad \text{(5.7 revisited)}$$

9.3.3.2.2 Assignment validity constraint function

With respect to the availability of developers, a solution is deemed feasible only if there are no conflicts in the assignment of each developer assigned to work on the project. For each individual x in the population, the evaluation is performed using the function previously defined in Equation (5.9), which sums the ratio of the number of time units with conflicts to the number of the total number of time units assigned for each developer r_j, to give the corresponding solution's degree of feasibility regarding this constraint.

$$\mathcal{G}_{\text{assignments}}(x) = \sum_{j=1}^{n} r_j^{\text{aviolations}}. \qquad \text{(5.9 revisited)}$$

9.4 Experiments

To help answer the research questions set concerning how well the algorithm performs in generating (near-)optimal solutions, two experiments were conducted.

The first experiment was carried out to help answer research question RQ5.1 by evaluating the performance of the algorithm in generating (near-)optimal solutions in the best-case scenario where all available developers possessing the highest experience levels also possessed the most suitable personality type for the corresponding profession.

The second experiment was executed to help answer research question RQ5.2 by evaluating the performance of the algorithm in generating (near-)optimal solutions in the worst-case scenario where all available developers possessing the highest experience levels also possessed the least suitable personality type for the corresponding profession, and vice-versa.

The experiments used two software project instances, $P5A$ and $P5B$, which were both created in a similar fashion as those used for experiments in Chapters 5 and 6. The characteristics of the two project instances are given in Table 9.2.

Table 9.2: Size and complexity of software projects used for experiments in Approach 5

Project instance	No. tasks	No. dependency relationships	Rate of dependency relationships	Avg. no. skills per task	No. available developers	Avg. no. skills per developer
$\boldsymbol{P5A}$	20	21	11%	2	10	3
$\boldsymbol{P5B}$	30	35	8%	2	12	4

The professions associated with each task of project $P5A$ are listed in Table 9.3. The specific professions used in the approach are taken from the Standard Occupational Classification (SOC) System [111], which allows for a formal categorization of the most common job positions found in the local software development industry.

Additionally, the duration and skill requirements of the tasks of project $P5A$, as well as the experience levels of the available developers are given in Tables 9.4 and 9.5, respectively. There are a 20 tasks in project $P5A$ requiring a total of 18 skills. The development company has ten resources possessing experience in one or more skills.

Since the aim is to assign developers to tasks that have already been scheduled, both project instances have a predefined plan for the execution of tasks. The Gantt chart in Figure 9.2 shows the predefined schedule of tasks for project $P5A$.

Table 9.3: Profession associated with each task of project $P5A$ used for experiments in Approach 5

Task	Profession	Task	Profession
$\boldsymbol{T01}$	Computer and information systems managers	$\boldsymbol{T11}$	Database architects
$\boldsymbol{T02}$	Computer and information systems managers	$\boldsymbol{T12}$	Database architects
$\boldsymbol{T03}$	Computer and information systems managers	$\boldsymbol{T13}$	Database architects
$\boldsymbol{T04}$	Computer and information systems managers	$\boldsymbol{T14}$	Database architects
$\boldsymbol{T05}$	Computer and information systems managers	$\boldsymbol{T15}$	Database architects
$\boldsymbol{T06}$	Computer systems analysts	$\boldsymbol{T16}$	Computer programmers
$\boldsymbol{T07}$	Computer systems analysts	$\boldsymbol{T17}$	Computer programmers
$\boldsymbol{T08}$	Computer systems analysts	$\boldsymbol{T18}$	Computer programmers
$\boldsymbol{T09}$	Computer systems analysts	$\boldsymbol{T19}$	Computer programmers
$\boldsymbol{T10}$	Database architects	$\boldsymbol{T20}$	Computer programmers

Table 9.4: Duration and skills required for the tasks of project $P5A$ used for experiments in Approach 5

		Required skill								
Task	**Duration**	$\boldsymbol{S01}$	$\boldsymbol{S02}$	$\boldsymbol{S03}$	$\boldsymbol{S04}$	$\boldsymbol{S05}$	$\boldsymbol{S06}$	$\boldsymbol{S07}$	$\boldsymbol{S08}$	$\boldsymbol{S09}$
$\boldsymbol{T01}$	18	✓	✓							
$\boldsymbol{T02}$	35	✓		✓						
$\boldsymbol{T03}$	29				✓					
$\boldsymbol{T04}$	17	✓	✓							
$\boldsymbol{T05}$	26			✓	✓					
$\boldsymbol{T06}$	13						✓			
$\boldsymbol{T07}$	12							✓	✓	
$\boldsymbol{T08}$	9							✓	✓	
$\boldsymbol{T09}$	3									✓

		Required skill								
Task	**Duration**	$\boldsymbol{S10}$	$\boldsymbol{S11}$	$\boldsymbol{S12}$	$\boldsymbol{S13}$	$\boldsymbol{S14}$	$\boldsymbol{S15}$	$\boldsymbol{S16}$	$\boldsymbol{S17}$	$\boldsymbol{S18}$
$\boldsymbol{T10}$	5	✓	✓							
$\boldsymbol{T11}$	14	✓	✓							
$\boldsymbol{T12}$	10	✓	✓							
$\boldsymbol{T13}$	12			✓						
$\boldsymbol{T14}$	9				✓					
$\boldsymbol{T15}$	16					✓				
$\boldsymbol{T16}$	7						✓			
$\boldsymbol{T17}$	23						✓	✓		
$\boldsymbol{T18}$	10						✓	✓		
$\boldsymbol{T19}$	3								✓	✓
$\boldsymbol{T20}$	9								✓	✓

Table 9.5: Experience levels of available developers in the skills required by the project $P5A$ used for experiments in Approach 5

	Required skill								
Developer	$S01$	$S02$	$S03$	$S04$	$S05$	$S06$	$S07$	$S08$	$S09$
$R01$	0.7364	0.8761		0.4298					
$R02$		0.3476	0.7066	0.5540	0.6286				
$R03$						0.6412	0.9959	0.7128	0.8696
$R04$						0.4323	0.7842	0.5784	0.2242
$R05$									
$R06$									
$R07$									
$R08$									
$R09$									
$R10$									

	Required skill								
Developer	$S10$	$S11$	$S12$	$S13$	$S14$	$S15$	$S16$	$S17$	$S18$
$R01$									
$R02$									
$R03$									
$R04$									
$R05$		0.6451		0.2819					
$R06$	0.9523	0.8780	0.3574	0.7090	0.1574				
$R07$						0.1026		0.0137	0.0105
$R08$						0.9472	0.5155	0.5393	0.4580
$R09$							0.0723	0.1530	0.2382
$R10$								0.8128	0.9531

The personality domain levels of the available developers in the two project instances for both experiments were specially initialized in a way so as to reflect the scenario being investigated. Specifically, for projects $P5A$ and $P5B$ in the first experiment, available developers that were assigned high levels of experience in the skills required by a task were also given personality domain levels suitable for the profession to which the task belonged, whereas available developers that were assigned low levels of experience in the skills required by a task were also given personality domain levels unsuitable for the profession to which the task belonged.

Conversely, for projects $P5A$ and $P5B$ in the second experiment, available developers that were assigned high levels of experience in the skills required by a task were also given personality domain levels unsuitable for the profession to which the task belonged, whereas available developers that were assigned low levels of experience in the skills required by a task were also given personality domain levels suitable for the profession to which the task belonged. Table 9.6

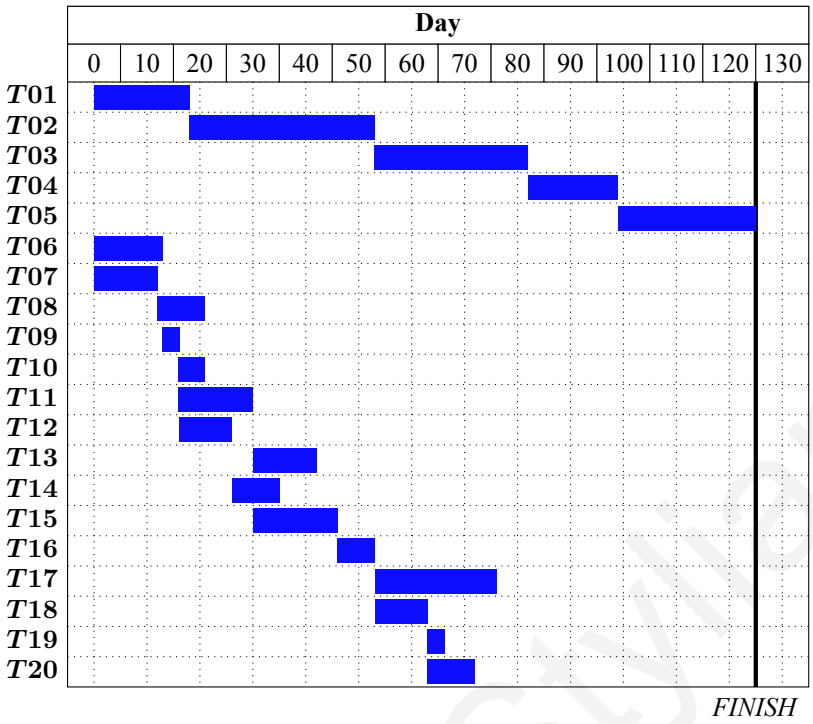


Figure 9.2: Gantt chart of project $P5A$ with predefined schedule of tasks used for experiments in Approach 5

Table 9.6: Personality domain levels of available developers in best-case and worst-case scenarios for project $P5A$ used for experiments in Approach 5

	Best-case scenario					**Worst-case scenario**				
Developer	**N**	**E**	**O**	**A**	**C**	**N**	**E**	**O**	**A**	**C**
***R*01**	Low	High	Average	Low	High	Low	High	Average	Low	High
***R*02**	Low	High	Average	Low	High	High	Low	Low	High	Low
***R*03**	Low	High	Average	High	Average	High	Low	High	Low	Low
***R*04**	High	Low	Average	Low	Average	Low	High	Average	High	Average
***R*05**	High	High	Low	High	High	Low	Low	High	Low	Low
***R*06**	Low	Low	High	Low	Low	High	High	Low	High	High
***R*07**	Average	Average	High	Average	High	Low	Low	Average	Low	Average
***R*08**	Low	Low	Average	Low	Average	High	High	High	High	Low
***R*09**	Average	Average	High	Average	High	Low	Low	Average	Low	Average
***R*10**	Low	Low	Average	Low	Average	High	High	Low	High	Low

The NSGA-II was run 10 times for each project instance in both experiments with a population of 100 individuals. The fast nondominated sorting procedure was applied to rank the individuals in terms of their fitness and feasibility, after which a tournament selection of size 4 was used to select which parents were to enter the mating pool. The best two parents were then recombined to

produce offspring using a one-point crossover operator with a likelihood of 0.80, and a single bit-flip mutation operator with a probability of $1/m$. The population evolved by repeating the steps from the selection of individuals until the termination criteria were met or the maximum number of iterations were reached (set at 2000 iterations).

9.5 Results and discussion

9.5.1 RQ5.1 How well does the algorithm perform in generating (near-)optimal solutions in cases where objectives are not at at all competing?

The first experiment involved executing the algorithm on the two project instances to evaluate the quality of the solutions generated with regards a best-case scenario where the objectives are not at at all competing. In such a scenario, only one (near-)optimal solution exists since in there will always be only one possible ideal assignment existing for each task. As expected, the algorithm was able to successfully generate this (near-)optimal solution in all ten runs for both project instances. Specifically, in both cases the algorithm managed to assign tasks to those developers possessing both high levels of experience and the most suitable desired personality type. Furthermore, the algorithm avoided assigning developers to a particular task if they were already assigned elsewhere at the same time. The (near-)optimal solution generated for project $P5A$ is shown in the resource allocation matrix in Table 9.7.

9.5.2 RQ5.2 How well does the algorithm perform in generating (near-)optimal solutions in cases where objectives are highly competing?

The second experiment involved executing the algorithm on the two project instances to evaluate the quality of the solutions generated with respect to a worst-case scenario where the objectives are highly competing. Here, it is anticipated that there will be many (near-)optimal solutions for each project, since the characteristics of developers (that is, experience levels and personality domain levels) were purposely set to be conflicting. Indeed, the algorithm for both project instances was able to generate a Pareto set of individuals that represent a number of different resource allocation solutions. By combining the final Pareto fronts of the ten runs, the approximation Pareto front for project $P5A$ consisted of a total of 94 unique solutions, whereas for project $P5B$ the approximation Pareto front consisted of a total of 95 unique solutions. The approximation Pareto front generated by the algorithm for project $P5A$ is shown in Figure 9.3.

Table 9.7: Resource allocation matrix of the (near-)optimal solution generated for project $P5A$ in best-case scenario in Approach 5

	Task									
Developer	$T01$	$T02$	$T03$	$T04$	$T05$	$T06$	$T07$	$T08$	$T09$	$T10$
$R01$	✓	✓		✓						
$R02$		✓	✓		✓					
$R03$						✓	✓	✓	✓	
$R06$										✓
$R08$										
$R10$										

	Task									
Developer	$T11$	$T12$	$T13$	$T14$	$T15$	$T16$	$T17$	$T18$	$T19$	$T20$
$R01$										
$R02$										
$R03$										
$R06$	✓	✓	✓	✓	✓					
$R08$						✓	✓	✓		
$R10$									✓	✓

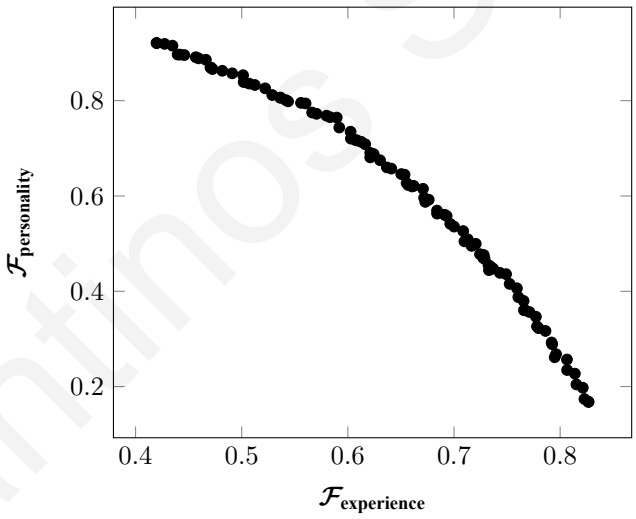


Figure 9.3: Approximation Pareto front corresponding to the best solutions generated by NSGA-II for project $P5A$ in Approach 5

In order to observe the trade-offs more clearly, the team size objective function was removed from the plot. As can be seen, there are many possible trade-offs between the levels of experience levels and the personality types of developers. Thus, a software project manager can select any one of the possible solutions.

9.6 Summary

Although there has been a vast number of studies investigating the personality type of software developers, there have been very little attempts in utilizing this knowledge into an applicable approach for resource-based software project scheduling, and in particular resource allocation. Importantly, this approach acts as a first step in actually incorporating this knowledge in an approach that can be of practical assistance to software project managers for the purpose of allocating resources to tasks efficiently and effectively through the use of multi-objective optimization and computational intelligence techniques.

One of the major contributions of this approach is that it can allow project managers to foresee possible resource issues arising during development. With respect to either or both constraint functions, if the algorithm is unable to find feasible solutions when applied to a specific project whose task schedule is predefined, this could indicate that the available resources are not sufficient or adequate enough to carry out the software project. This is very useful for project managers since it would allow them to revise their task schedule and attempt to allocate their resources in a slackened time frame. Alternatively, without modifying the project's schedule, a project manage may use the results to recruit extra resources (possessing either higher experience levels or more suitable personality traits or both). In a similar way, the approach can be used to examine whether the development company has the required capacity in terms of human resources before bidding for a software project.

Chapter 10

Conclusions and Future Research

10.1 Summary

This dissertation has presented the research work carried out in the area of resource allocation and task scheduling in software development projects. The main goals of the research were to investigate these activities in order to identify the major challenges and open issues, and to develop a number of approaches to help solve the problem using single or multiple objective optimization methods.

Resource allocation and task scheduling activities are considered two of the most important planning activities that project managers are required to undertake because they must be carried out properly at the start of a project. Failure to do so can potentially jeopardize the success of a project by causing schedule delays and budget overruns or a decline in quality. However, due to the lack of proper practices, in addition to the computationally-intensive nature of these activities, it is difficult for project managers to accurately and efficiently carry out these tasks. For this reason, many researchers in the field of software engineering have approached the problem in the context of operational research, treating it as a special instance of the resource-constrained project scheduling problem, which can be solved using a variety of optimization methods. A comprehensive review of a number of research approaches that have previously attempted to solve the problem in this manner was presented in Chapter 3.

The approaches developed employed multiobjective optimization methods as the means to help software project managers simultaneously assign tasks to developers and plan the execution of tasks in a way that, on the one hand, meets their objectives and, on the other hand, satisfies their

constraints. The approaches presented in Chapters 4–7 were specifically developed to deal with the problem of resource allocation and task scheduling mainly using technical aspects related to software development, such as the required skills, cost and duration of tasks, as well as the level of experience and productivity rates of developers. Each developed approach defined and represented the problem accordingly so that these aspects were incorporated in the corresponding optimization method either as objectives or constraints responsible for generating optimal and feasible resource allocation and task scheduling solutions. The series of experiments conducted to validate each approach along with the subsequent results were also provided so as to demonstrate the effectiveness of each approach. Experiments used a number of generated software project instances that were specially constructed with the help of local software project managers in the aim of testing the developed approaches using projects whose task characteristics and resource attributes depicted more accurate representations of real-world software projects. Apart from generated software project instances, experiments carried out in Approach 4 also used data from a real-world software project provided by a local software development company.

The main goal of the first three approaches was to allocate resources and schedule tasks so as to minimize the duration of software projects and to maximize the experience of the developers assigned. From the initial results obtained in these three approaches, it was clear that the optimization methods were able to successfully generate feasible and optimal resource allocation and task scheduling solutions for smaller-sized projects. However, for larger-sized projects the findings indicated that generated solutions severely lacked sufficient quality, both in terms of optimality and feasibility. In order to overcome these limitations, each new approach developed gradually improved the previous ones by including new or modifying existing objective and constraint functions.

Approach 4 (Chapter 7), in particular, introduced the most important improvements, such as a new representation of individuals to improve the quality of solutions, the addition of an objective to manage project cost and the inclusion of productivity-based attributes of developers to allow for more useful and accurate duration and cost estimates. Several experiments were carried out to evaluate the performance of four well-known multiobjective genetic algorithms (MOCell, NSGA-II, PAES, and SPEA2) using 16 generated software project instances. From the analysis of the results obtained, it was clear that MOCell, NSGA-II and SPEA2 were the most dominant of the four algorithms, managing to outperform PAES in the majority of project instances. This suggested that

for this approach, these algorithms are better for providing resource allocation and task scheduling solutions that were closer to the optimal, as well as more diverse. Additionally, 36 project instances were used to compare the scalability of the algorithms. Here, the results indicated that PAES was able to scale better than the rest, despite producing solutions with lower quality. Overall, there were positive indications that this approach could help project managers make better resource allocation and task scheduling decisions, since the approach was based on more realistic assumptions and used more representative information.

Furthermore, the results from the experiment on the real-world software project provided evidence that the approach could also provide a basis for a promising decision-support tool with which software project managers could be able to efficiently select a suitable allocation of resources and task schedule that satisfies his/her criteria the most from a range of alternatives through the use of multiobjective optimization.

Since human resources are considered the most crucial type of resource in software development, it is important for project planning decisions to also take into account human aspects of software development. The personality types and preferences of software developers is one such aspect, and was examined as part of the research with respect to resource allocation and team formation. Chapter 8 presented various studies that illustrated the importance of personality type in software development. Some of these studies focused simply on assessing the personality type of developers in various software professions in order to form a personality profile of each developer. Other studies used personality type in resource allocation approaches by assigning developers to tasks that they are best suited for. Finally, a number of studies assessed personality types in attempt to discover correlations between team performance and the personality type of team members so as to improve the effectiveness of team formation.

Following the investigation of personality type, Chapter 9 presented an approach that employed a Pareto ranking genetic algorithm for allocating resources to tasks using both technical and human aspects. Specifically, this approach used an optimization function that aimed to: (1) maximize the experience of assigned developers, (2) maximize the suitability of assigned developers, and (3) minimize the number of assigned developers in each task. The suitability of a developer is determined by measuring the distance between a developer's personality type and the desired personality of the profession to which the task belongs. The desired personality type of each software profession was identified by corresponding different personality traits related to the domains

of the FFM [125, 126] to the job-related and worker-related characteristics and requirements provided by O*Net [112]. In this way, the smaller the distance between the actual personality (of the developer) and the target personality (of the task), then the more suitable the developer is to undertake the tasks. Again, experiments were carried out to assess the performance of the algorithm using a number of software project instances. The results obtained from the experiments indicated that the algorithm was capable of generating optimal and feasible solutions in the majority of runs executed. In the cases where the attributes of the developers did not cause the objectives to be competing (that is, in cases where the most experience developer for a task was also the most suitable in terms of personality), the algorithm correctly managed to generate the one optimal solution performing the best assignment possible. In the cases where the attributes of the developers did cause the objectives to be competing (that is, in cases where the most experience developer for a task was also the least suitable in terms of personality), the algorithm attempted to balance the two objectives accordingly so as to generate a set of optimal solutions consisting of different trade-offs between experience and personality.

Furthermore, the approaches could also provide a basis for a promising decision-support tool with which software project managers are able to efficiently select a suitable allocation of resources and task schedule that satisfies his or her criteria the most from a range of alternatives through the use of multiobjective optimization. Multi-objective optimization allows for different trade-offs between duration and cost to be examined by software project managers, which would otherwise not be possible due to the many permutations that require effort and time to produce manually. It is important, however, to examine the approaches using more real-world projects from the local software industry. To this end, several development companies have been contacted to provide project and resource data for further experimentation.

10.2 Contributions and benefits

10.2.1 Task characteristics and Resource attributes

A key feature that distinguishes this research from other previous attempts is the fact that the objectives and constraints specified in each developed approach take into account various software development-related factors that are more realistic in reflecting the criteria actually considered by project managers in the software development industry. Specifically, these software development-related factors concern the characteristics of the tasks (including required effort and skills, and

interdependence type) and attributes of the resources (such as experience, productivity, salary and personality). Consequently, the proposed approaches offer the benefit of being able to provide project managers with resource allocation and tasks scheduling solutions that are more accurate as they are assessed using more realistic criteria. Hence, the contributions of the proposed approaches are a direct result of the inclusion of these task characteristics and resource attributes.

10.2.1.1 Developer experience (Approaches 1–3 and 5)

The inclusion of this resource attribute is so that resource allocation and task scheduling activities take in to account the fact that developers possessing a certain skill are not necessarily equally capable of carrying out a task requiring that skill. In reality, developers should be considered noninterchangeable since each developer is likely to have a different level of experience in each skill. Thus, the assignment of a less experienced developer to a task will have a different effect than the assignment of a more experienced one. Thus, by taking into account developers' experience, the solutions generated by these proposed approaches will better reflect the effect that the noninterchangeable nature of developers has on the allocation of resources.

10.2.1.2 Productivity rate of developers (Approach 4)

This resource attribute is incorporated in order for resource allocation and task scheduling activities to adequately reflect the fact that the duration of a task is a direct result of the relationship between the effort required to complete a task and the productivity rate of the developer assigned to carry it out; a more productive developer will be able to complete a task within a shorter duration than a less productive developer. Therefore, by considering the productivity rate of developers, task durations are calculated more realistically and so the proposed approach provides project managers with solutions where the execution of tasks are planned more accurately.

10.2.1.3 Task interdependence type (Approach 4)

The addition of this task characteristic is for resource allocation and task scheduling activities to be able to deal properly with the common problem of how developers combine their efforts when assigned to work together in different types of software tasks, since this, in turn, affects the way that task durations are computed. In particular, software tasks are not all carried out in the same way because the work that needs to done depends on the desired output of each task. This means

that developers cannot always combine their efforts (that is, their productivity) in the same manner for all tasks. Consequently, by using the interdependence type of tasks, the proposed approach offers the ability to determine how developers will combine their productivity rates when working on different tasks in order to calculate more realistic task durations (when more than one developer is assigned to a task). As a result, solutions formed by the proposed approach will contain more precise task schedules since they will be based on more accurate task durations.

10.2.1.4 Communication overhead (Approach 4)

The inclusion of this task characteristic is so that resource allocation and task scheduling activities factor in the additional time that results from the need for developers to communicate, collaborate and cooperate with each other during the execution of a task. Communication, while conducive in order for a task to be carried out, is not considered as part of the effort required to perform it. In this regard, the duration of a task needs to be adapted accordingly in order to accommodate the communication overhead incurred during its execution. Despite being a characteristic of tasks, it is actually calculated based on the number of resources assigned. By adjusting the durations of tasks through the addition of communication overhead in the proposed approach, generated solutions will again comprise more accurate schedules.

10.2.1.5 Personality types of software developers (Approach 5)

Apart from including technical factors in resource allocation and task scheduling activities, it is equally important to consider nontechnical factors. Human factors, especially, are significant since the resources available for software development companies consist to a large degree, if not wholly, of human resources. One of these factors involves personality and, in particular, the personality type of software professionals. This developer attribute is taken into account in order for resource allocation and task scheduling activities to help project managers mitigate issues such as underperformance and low productivity, which result from the assignment of developers to tasks that they are less suited to carry out in terms of personality type. The approach developed assesses the suitability of developers by matching their personality type to that desired by the software professions. Thus, incorporating how suited a developers is to a task based on personality can provide project managers with solutions containing better allocation of resources.

10.2.2 Pareto ranking optimization

The benefit of employing multiobjective optimization methods that use Pareto ranking is their ability to generate a set of solutions, rather than just a single one. In this way, approaches such as the ones described in Chapters 6, 7 and 9 are able to provide software project managers with a selection of alternative resource allocation and task scheduling solutions to choose from. If the criteria of a project manager favours one of the objectives (for instance, to allocate resources and schedule tasks so that project duration is minimized), then the solution chosen will be from ones whose project duration objective function value corresponds to shorter make spans. Alternatively, if a project manager's goal is to allocate resources and schedule tasks so that the developers assigned possess the greatest level of experience, then the selection would be made from solutions with higher values in the developer experience objective function. A more common option, however, would be to choose a solution that balances the criteria.

10.3 Limitations

The developed approaches made use of optimization methods, found in the field of computational intelligence, aiming to satisfy a number of criteria and conditions. Because of certain properties of the algorithms employed, the approaches can be subject to several limitations, particularly from threats to the validity of the appropriateness of the assumptions made, the experimental process carried out to obtain results and the generalized inferences made from these results.

10.3.1 Construct validity

Threats to construct validity concern the assumptions and simplifications made in the developed approaches regarding the software development process. The main assumption in all of the approaches was that developers are only allowed to be assigned to one task at any given time in a project. They did not take into account the possibility of having developers simultaneously assigned to multiple tasks with certain degrees of dedication. However, taking this into account was not in the scope of the current research work. Regardless, to accommodate this, minimal changes will be require to the way that the feasibility of individuals is calculated (in all approaches) and to the way that tasks are scheduled (in Approach 4). Also, an assumption is made in Approach 4 regarding the maximum rate of productivity of developers (set at value of 2.0), which in practice may

never be reached or exceeded. Nevertheless, any potential effect of this threat is limited, since the productivity rate values used in the approach scale in relative terms rather than in absolute terms.

10.3.2 Internal validity

Optimization methods such as genetic algorithms and particle swarm optimization algorithms are stochastic in nature. They use various degrees of randomness during the initialization and evolution of populations/swarms so that they can generate solutions. In order to limit this internal validity threat in the approaches, each optimization method employed was run a number of times for each project instance investigated. Furthermore, given the random nature of the optimization methods, the results of the experiments may be influenced by the parameters chosen for each optimization method. So, in order to mitigate this in each approach, preliminary runs were carried out using different settings to help identify the best parameters with which to execute the experiments. Moreover, the experiments carried out in Approach 4 employed statistical tests to examine if significant differences existed between the results generated by each algorithm examined in the approach.

10.3.3 External validity

The main threat to external validity is the fact that experiments conducted in the approaches were carried out using randomly-generated project instances, which therefore limits the ability to make generalizations from the findings. To mitigate this threat, the project instances were created through discussions with several local software project managers to help extract various task and developer features, including number of tasks, complexity of task dependencies, number of available developers and salary ranges. These features were then used to randomly generate the project instances as realistic as possible. In Approach 4, an experiment using a real-world case study was also conducted, which showed that the solutions generated were of better quality and more realistic than compared to the resource allocation and task schedule constructed manually by the software project manager, let alone the fact that these solutions were generated much quicker. However, further experiments using real-world software projects are necessary in order to support the results already obtained, and are planned as part of future research. In addition, the ability of the optimization methods to achieve a satisfactory level of quality of solutions in a reasonable amount of time is dependent on the number of iterations that they are left to run for. This means that

for larger software projects, there may be an issue of scaling, where the optimization method will require longer computational time to find better solutions. This threat, however, was addressed by preliminarily running the algorithms with different iteration settings, before finally selecting an appropriate number in each developed approach. From the findings, it was concluded that significant improvements to the results were not expected by increasing the number of iterations further than the ones reported. Computation overhead may be addressed even more efficiently in cases of large projects using modules in high performance computing environments, thus executing in much less time compared to the original experiments.

10.4 Recommendations for further research

There are a number of potential topics that can be explored as part of future research work. These topics focus on ways to improve the current approaches and, also, on ways to utilize personality for resource allocation and task scheduling, as well as team formation in other approaches.

Regarding the improvement of approaches, one possible area for further study concerns the better understanding of developer productivity in order to help provide more accurate estimations of the duration and cost of tasks. Since the topic of productivity is very broad, a thorough investigation will be required to ensure that its usage in the optimization methods contributes to the generation of realistic solutions. Also, there is a need to explore the rates of completion of software development tasks at different levels of productivity, and to examine whether a saturation point exists at which a task cannot be completed quicker irrespective of the rate of productivity.

One of the objectives in the approaches described in Chapters 4-6 and Chapter 9 is to maximize the level of experience of the assigned developers. However, some tasks may require skills at a specific level of experience that is not necessarily the maximum level of experience. Some trivial tasks, for example, may not require skills at a high level of experience and, consequently, can be assigned to developers with lower levels of experience in the corresponding required skills. An enhancement in these approaches could involve the addition of lower and upper bounds to the level of experience in a required skill so as to avoid wasting resources, that is, the assignment of tasks to overqualified developers, which may be costly for development companies. Furthermore, by only assigning highly-experienced developers to tasks may cause the exclusion of less-experienced developers from being selected to work on the software project. Introducing lower and upper

bounds should help avoid this as more developers will have the opportunity to be assigned to tasks.

The approaches currently assume that resources are only permitted to work on one task at any given time. However, this is not always the case in reality since in most development companies developers may be required to split their time between multiple tasks possibly belonging to multiple projects. Future work could, therefore, focus on incorporating degrees of resource dedication and availability, as well as resource levelling constraints for multiproject resource allocation and task scheduling. In addition, it would also be interesting to research the allocation of resources and scheduling of tasks for distributed software development projects, where available resources are located around the world and use the principle of follow-the-sun development to build software. An attempt has already been started based on the approach described in Chapter 7, with the aim of extending the approach so as to allocate resources and schedule tasks of multiple projects with developers from multiple geographical locations.

The approaches aim to help software project managers to assign developers to tasks and plan the execution of tasks at the start of a project. However, it is equally important to provide software project managers a way to handle the reassignment of developers and rescheduling of tasks once the project has commenced. In real-world software projects, many events take place during the course of a project, both within and beyond the control of the development company, including the introduction of new tasks brought on by new requirements and designs, as well as the departure of developers from the project or from the development company because of illness, transfers or resignations. A future improvement should provide a reassignment and rescheduling mechanism to the approaches described so as to allow software project managers allocate resources and schedule tasks dynamically as the project is proceeding.

Further research will also be required to determine the interdependence type of software development tasks so as to be able to estimate precisely how developers will combine their efforts for each task, and subsequently, to be able to calculate the duration of each task, accordingly. Furthermore, research will also be necessary to handle the release of developers working on divisible tasks. Specifically, in divisible tasks (additive or conjunctive), the work is broken down into subtasks, and then the subtasks are divided among the team members. Ideally, once a developer completes his/her subtask, they should be made available for assignment again since their contribution to the task is over, rather than have him/her committed for the full duration of the task.

As regards the optimization methods, further experiments should be carried out using other genetic algorithm variations, as well as other alternative optimization techniques, both population-based and otherwise, in order to determine whether such methods are able to provide more diverse and/or better resource allocation and task scheduling solutions. In addition, another area that should be looked into concerns the representation of solutions. The approaches currently use a representation whose solution space contains both feasible and infeasible solutions. Further research should examine more effective representations that lead to a search space that creates only feasible solutions to reduce the number of infeasible solutions generated. Finally, the optimization methods should be incorporated into a decision-support tool for software project managers to use, providing them with the ability to select the desired objectives and constraints for resource allocation and task scheduling, as well as the optimization method and related execution parameters.

With respect to personality, additional research is required to further analyze the desired personality type of software professions so that a more complete profile of what is required of a profession in terms of personality can be established. The approach currently adopts the FFM personality model for assessing the suitability of developers; though, other instruments could also be investigated, such as the MBTI [122] or the KTS [124].

Moreover, personality type can be used to further enhance Approach 4 by incorporating the developer personality objective function used in Approach 5, in the optimization function of Approach 4. This will allow the optimization method employed to allocate resources and schedule tasks in a way that also maximizes the suitability of assigned developers (in addition to minimizing the cost and duration of the project).

One other possible research topic could involve developing an approach that uses personality types and preferences of software developers in order to help software project managers form highly effective development teams in terms of performance. As discussed in Chapter 8, there have been many studies carried out examining the possible correlation between personality type and performance. Thus, it would be highly beneficial for a software project manager to be able to predict whether a specific team chosen for a project will be successful based not only on the team member's technical skills and knowledge, but also on their personality type. Research in this topic has already begun with the extension of a fuzzy cognitive map model [154] proposed by Andreou and Neophytou [155] depicting the various concepts affecting team effectiveness (including personality types), and the interactions between these concepts. The goal of the research will be to

initialize the concepts of the model with the characteristics of the team members, and then to simulate the behaviour of these concepts in order to predict whether these team member characteristics will interact in a way that leads to project success. Additionally, since 2012, the NEO-FFI-3 [125] questionnaire has been administered to students enrolled in a project-oriented Software Engineering university course for the purpose of investigating possible correlations between personality and software-related aspects, such as performance, job satisfaction, team cohesion and software quality. Due to the small number of teams participating each year, there has not been enough data acquired for any general and reasonable inferences to be made as yet. Results of this research will require several more years to report once enough data has been collected from teams.

An important part of this research topic should also involve investigating how individual personality type is combined into a team personality type, similarly to the way that productivity is handled with task interdependence type in Approach 4. It may not be possible to simply average the personality types of team members in order to determine the overall team personality type. Therefore, a thorough study will be required to determine how to aggregate individual personality traits.

10.5 Concluding remarks

The problem of resource allocation and task scheduling, particularly in software development projects, is difficult to solve due to its multidisciplinary nature. It encompasses the fields of software engineering, management and industrial organization, and even personality psychology. Due to the inherent attributes of software, incorrect decisions in these activities may not be detected on time, which could prove catastrophic for a software development company, especially in projects for the development of large, complex multiproduct software systems. Furthermore, software developers are knowledge workers, and it is with this knowledge that software is built. As a result, it is very hard to quantify the effect that this knowledge has in various aspects of software development, such as duration and quality. The approaches developed and presented in this dissertation have made a first attempt to quantify this effect by representing knowledge, first, in the form of experience and, second, in the form of productivity.

Traditional approaches to solving resource allocation and task scheduling make use of various modelling techniques and intelligence-based methods based on technical project criteria, such as project cost and duration, required skills, or number of defects. However, research in the field

has started to move towards a more human-centric solution, with an increase in the number of approaches including nontechnical aspects of software development, such as personality types and preferences. One of the biggest challenges, therefore, for the research community is to try to find a way to 'marry' these two methods of solving the problem. The ideal direction for research would be to concentrate on providing a hybrid of the two styles in a unified software project resource allocation and task scheduling framework that, on the one hand, takes advantage of the benefits of underlying techniques (mathematical modelling/computational intelligence) and, on the other hand, targets both technical/industrial objectives and nontechnical/human-centric criteria. One of the obstacles to achieving this is quantifying and measuring human-centric criteria. The approaches developed as part of the research work reported in this dissertation have already attempted to contribute to solving this problem.

Concluding, this particular area of research is very promising as it contributes to dealing with the important issue of helping software projects succeed by focusing on the most crucial, and arguably only, resource involved in software development.

Bibliography

[1] The Standish Group International, "CHAOS Report 2015," tech. rep., Apr. 2015.

[2] S. T. Acuña, N. Juristo, A. M. Moreno, and A. Mon, *A Software Process Model Handbook for Incorporating People's Capabilities*. Secaucus, NJ, USA: Springer US, 2005.

[3] D. Varona, L. F. Capretz, Y. Piñero, and A. Raza, "Evolution of software engineers' personality profile," *ACM SIGSOFT Software Engineering Notes*, vol. 37, pp. 1–5, Jan. 2012.

[4] G. A. Neuman, S. H. Wagner, and N. D. Christiansen, "The relationship between work-team personality composition and the job performance of teams," *Group and Organization Management*, vol. 24, pp. 28–45, Mar. 1999.

[5] D. J. Reifer, ed., *Software Management*. Hoboken, NJ, USA: Wiley-IEEE Computer Society, 7th ed., 2006.

[6] J. Gouws and L. E. Gouws, *Fundamentals of Software Engineering Project Management*. Perth, Australia: Melikon Pty, 2004.

[7] "Systems and Software Engineering – Software Life Cycle Processes." ISO/IEC/IEEE Std. 12207-2008, 2008.

[8] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "Capability maturity model^{SM} for software, version 1.1," Tech. Rep. CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Feb. 1993.

[9] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. Boston, MA, USA: McGraw-Hill, 5th ed., 2001.

[10] D. J. Reifer, "Software management's seven deadly sins," *IEEE Software*, vol. 18, pp. 12–15, Mar.–Apr. 2001.

[11] W. Curtis, W. Hefley, and S. Miller, "People capability maturity model (P-CMM) Version 2.0, second edition," Tech. Rep. CMU/SEI-2009-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Jul. 2009.

[12] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ, USA: Prentice Hall, 1981.

[13] A. B. Pyster and R. H. Thayer, "Software engineering project management 20 years later," *IEEE Software*, vol. 22, pp. 18–21, Sep.–Oct. 2005.

[14] W. S. Humphrey, "Why big software projects fail: The 12 key questions," *CrossTalk: The Journal of Defense Software Engineering*, vol. 18, pp. 25–29, Mar. 2005.

[15] S. McConnell, "The nine deadly sins of project planning," *IEEE Software*, vol. 18, pp. 5–7, Sep.–Oct. 2001.

[16] P. E. McMahon, "Bridging agile and traditional development methods: A project management perspective," *CrossTalk: The Journal of Defense Software Engineering*, vol. 17, pp. 16–20, May 2004.

[17] F. Ahlemann, "Towards a conceptual reference model for project management information systems," *International Journal of Project Management*, vol. 27, pp. 19–30, Jan. 2009.

[18] J. S. Reel, "Critical success factors in software projects," *IEEE Software*, vol. 16, pp. 18–23, May-Jun. 1999.

[19] L. Raymond and F. Bergeron, "Project management information systems: An empirical study of their impact on project managers and project success," *International Journal of Project Management*, vol. 26, pp. 213–220, Feb. 2008.

[20] T. McBride, "The mechanisms of project management of software development," *Journal of Systems and Software*, vol. 81, pp. 2386–2395, Dec. 2008.

[21] R. H. Thayer, A. Pyster, and R. C. Wood, "The challenge of software engineering project management," *IEEE Computer*, vol. 13, pp. 51–59, Aug. 1980.

[22] J. Rothman, "Successful software management: 14 lessons learned," *CrossTalk: The Journal of Defense Software Engineering*, vol. 16, pp. 17–20, Dec. 2003.

[23] R. E. Kraut and L. A. Streeter, "Coordination in software development," *Communications of the ACM*, vol. 38, pp. 69–81, Mar. 1995.

[24] J. Tucker, A. Mackness, and H. Rutledge, "The human dynamics of IT teams," *CrossTalk: The Journal of Defense Software Engineering*, vol. 17, pp. 15–19, Feb. 2004.

[25] R. Thomsett, "The clash of two cultures: Project management versus process management," *American Programmer*, vol. 7, pp. 18–28, Jun. 1994.

[26] F. P. Brooks Jr, "No silver bullet: Essence and accidents of software engineering," *IEEE Computer*, vol. 20, pp. 10–19, Apr. 1987.

[27] M. Corporation, "Project®project and portfolio management software," 2013. Available at https://products.office.com/en/project/project-and-portfolio-management-software.

[28] I. Experience in Software, "Project KickStart™ 5," 2008. Available at https://www.projectkickstart.com/products/project_kickstart.cfm.

[29] Basecamp, "Basecamp 3," 2015. Available at https://basecamp.com/.

[30] MatchWare®, "MindView™ 6," 2015. Available at http://www.matchware.com/en/products/mindview/default.htm.

[31] Stand By Soft Srl, "RationalPlan MultiProject," 2016. Available at http://www.rationalplan.com/multi-project-management-software.php.

[32] S. Petter and V. Vaishnavi, "Facilitating experience reuse among software project managers," *Information Sciences*, vol. 178, pp. 1783–1802, Apr. 2008.

[33] R. T. Hughes, *Project Management Software*, pp. 139–153. Encyclopedia of Physical Science and Technology, New York, NY, USA: Academic Press, 3rd ed., Oct. 2003.

[34] C. K. Chang, M. J. Christensen, and T. Zhang, "Genetic algorithms for project management," *Annals of Software Engineering*, vol. 11, pp. 107–139, Nov. 2001.

[35] C. K. Chang, H.-Y. Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-line based model for software project scheduling with genetic algorithms," *Information and Software Technology*, vol. 50, pp. 1142–1154, Oct. 2008.

[36] N.-H. Pan, P.-W. Hsaio, and K.-Y. Chen, "A study of project scheduling optimization using tabu search algorithm," *Engineering Applications of Artificial Intelligence*, vol. 21, pp. 1101–1112, Oct. 2008.

[37] A. Heiat, "Comparison of artificial neural network and regression models for estimating software development effort," *Information and Software Technology*, vol. 44, pp. 911–922, Dec. 2002.

[38] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," *Empirical Software Engineering*, vol. 9, pp. 229–257, Sep. 2004.

[39] C. Stylianou and A. S. Andreou, "A hybrid software component clustering and retrieval scheme using an entropy-based fuzzy k-modes algorithm," in *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1, pp. 202–209, IEEE Computer Society, Oct. 2007.

[40] C. C. Michael, G. E. McGraw, M. A. Schatz, and C. C. Walton, "Genetic algorithms for dynamic test data generation," in *Proceedings of the 12th IEEE International Conference on Automated Software Engineering*, pp. 307–308, IEEE Computer Society, Nov. 1997.

[41] L. V. Kantorovich, "A new method of solving of some classes of extremal problems," *Doklady Akademii Nauk SSSR*, vol. 28, pp. 211–214, May 1940.

[42] C. Li, J. M. van den Akker, S. Brinkkemper, and G. Diepen, "Integrated requirement selection and scheduling for the release planning of a software product," in *Requirements Engineering: Foundation for Software Quality* (P. Sawyer, B. Paech, and P. Heymans, eds.), vol. 4542 of *Lecture Notes in Computer Science*, pp. 93–108, Berlin, Germany: Springer-Verlag, Jun. 2007.

[43] L. D. Otero, G. Centeno, A. J. Ruiz-Torres, and C. E. Otero, "A systematic approach for resource allocation in software projects," *Computers and Industrial Engineering*, vol. 56, pp. 1333–1339, May 2009.

[44] A. Ejnioui, C. E. Otero, and L. D. Otero, "A multi-attribute decision making approach for resource allocation in software projects," in *Proceedings of the 2012 International Conference on Software Engineering Research and Practice* (H. R. Arabnia, H. Reza, and J. Xiong, eds.), pp. 12–16, CSREA Press, Jul. 2012.

[45] C. E. Otero, L. D. Otero, I. Weissberger, and A. Qureshi, "A multi-criteria decision making approach for resource allocation in software engineering," in *Proceedings of the 12th International Conference on Computer Modelling and Simulation (UKSIM 2010)*, pp. 137–141, IEEE Computer Society, Mar. 2010.

[46] H.-T. Tsai, H. Moskowitz, and L.-H. Lee, "Human resource selection for software development projects using Taguchi's parameter design," *European Journal of Operational Research*, vol. 151, pp. 167–180, Nov. 2003.

[47] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[48] O. Bellenguez and E. Néron, "Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills," in *Practice and Theory of Automated Timetabling V* (E. Burke and M. Trick, eds.), vol. 3616 of *Lecture Notes in Computer Science*, pp. 229–243, Berlin, Germany: Springer-Verlag, Aug. 2004.

[49] O. Bellenguez-Morineau and E. Néron, "A branch-and-bound method for solving multi-skill project scheduling problem," *RAIRO Operations Research*, vol. 41, pp. 155–170, Apr.–Jun. 2007.

[50] F. Padberg, "Scheduling software projects to minimize the development time and cost with a given staff," in *Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC 2001)*, pp. 187–194, IEEE Computer Society, Dec. 2001.

[51] F. Padberg, "Using process simulation to compare scheduling strategies for software projects," in *Proceedings of the 9th Asia-Pacific Software Engineering Conference (APSEC 2002)*, pp. 581–590, IEEE Computer Society, Dec. 2002.

[52] F. Padberg, "A software process scheduling simulator," in *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pp. 816–817, IEEE Computer Society, May 3-10 2003.

[53] F. Padberg, "Computing optimal scheduling policies for software projects," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, pp. 300–308, IEEE Computer Society, Nov. 30–Dec. 3 2004.

[54] F. Padberg, "A study on optimal scheduling for software projects," *Software Process: Improvement and Practice*, vol. 11, pp. 77–91, Jan.–Feb. 2006.

[55] G. Antoniol, A. Cimitile, G. A. D. Lucca, and M. D. Penta, "Assessing staffing needs for a software maintenance project through queuing simulation," *IEEE Transactions on Software Engineering*, vol. 30, pp. 43–58, Jan. 2004.

[56] P. Jalote and G. Jain, "Assigning tasks in a 24-hour software development model," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, pp. 309–315, IEEE Computer Society, Nov. 30–Dec. 3 2004.

[57] A. Barreto, M. de Oliveira Barros, and C. M. L. Werner, "Staffing a software project: A constraint satisfaction approach," in *Proceedings of the 7th International Workshop on Economics-driven Software Engineering Research (EDSER 2005)*, pp. 1–5, ACM, May 2005.

[58] A. Barreto, M. de O. Barros, and C. M. L. Werner, "Staffing a software project: A constraint satisfaction and optimization-based approach," *Computers and Operations Research*, vol. 35, pp. 3073–3089, Oct. 2008.

[59] C. K. Chang, C. Chao, S.-Y. Hsieh, and Y. Alsalqan, "SPMNet: A formal methodology for software management," in *Proceedings of the 18th Annual International Computer Software and Applications Conference (COMPSAC 1994)*, p. 57, IEEE Computer Society, Nov. 9-11 1994.

[60] C. K. Chang, C. Chao, T. T. Nguyen, and M. J. Christensen, "Software project management net: A new methodology on software management," in *Proceedings of the 22nd Annual International Computer Software and Applications Conference (COMPSAC 1998)*, pp. 534–539, IEEE Computer Society, Aug. 1998.

[61] Y. Ge and C. K. Chang, "Capability-based project scheduling with genetic algorithms," in *Proceedings of the 2006 International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA-IAWTIC 2006)*, p. 161, IEEE Computer Society, Nov. 28–Dec. 1 2006.

[62] H. Jiang, C. K. Chang, J. Xia, and S. Cheng, "A history-based automatic scheduling model for personnel risk management," in *Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, vol. 2, pp. 361–366, IEEE Computer Society, Jul. 2007.

[63] Y. Ge, "Software project rescheduling with genetic algorithms," in *Proceedings of the 2009 International Conference on Artificial Intelligence and Computational Intelligence (AICI 2009)*, vol. 1, pp. 439–443, IEEE Computer Society, Nov. 2009.

[64] E. Alba and J. F. Chicano, "Management of software projects with GAs," in *Proceedings of the 6th Metaheuristics International Conference (MIC 2005)*, pp. 13–18, Aug. 2005.

[65] E. Alba and J. F. Chicano, "Software project management with GAs," *Information Sciences*, vol. 177, pp. 2380–2401, Jun. 2007.

[66] F. Luna, D. L. Gonzalez-Alvarez, F. Chicano, and M. A. Vega-Rodriguez, "On the scalability of multi-objective metaheuristics for the software scheduling problem," in *Proceedings of the 11th International Conference on Intelligent Systems Design and Applications*, pp. 1110–1115, IEEE Computer Society, Nov. 2011.

[67] F. Chicano, F. Luna, A. J. Nebro, and E. Alba, "Using multi-objective metaheuristics to solve the software project scheduling problem," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO 2011)*, pp. 1915–1922, ACM, Jul. 2011.

[68] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, pp. 308–320, Dec. 1976.

[69] J. Duggan, J. Byrne, and G. J. Lyons, "A task allocation optimizer for software construction," *IEEE Software*, vol. 21, pp. 76–82, May-Jun. 2004.

[70] P. Kapur, A. Ngo-The, G. Ruhe, and A. Smith, "Optimized staffing for product releases and its application at Chartwell Technology," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, pp. 365–386, Sep. 2008.

[71] A. Ngo-The and G. Ruhe, "Optimized resource allocation for software release planning," *IEEE Transactions on Software Engineering*, vol. 35, pp. 109–123, Jan.–Feb. 2009.

[72] G. Antoniol, M. D. Penta, and M. Harman, "Search-based techniques for optimizing software project resource allocation," in *Genetic and Evolutionary Computation – GECCO 2004* (K. Deb, ed.), vol. 3103 of *Lecture Notes in Computer Science*, pp. 1425–1426, Berlin, Germany: Springer-Verlag, Jun. 2004.

[73] G. Antoniol, M. D. Penta, and M. Harman, "Search–based techniques applied to optimization of project planning for a massive maintenance project," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005)*, pp. 240–249, IEEE Computer Society, Sep. 2005.

[74] S. Gueorguiev, M. Harman, and G. 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 (GECCO 2009)*, pp. 1673–1680, ACM, Jul. 2009.

[75] M. D. Penta, M. Harman, and G. Antoniol, "The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study," *Software: Practice and Experience*, vol. 41, pp. 495–519, Apr. 2011.

[76] J. Ren, M. Harman, and M. D. Penta, "Cooperative co-evolutionary optimization of software project staff assignments and job scheduling," in *Search Based Software Engineering* (M. B. Cohen and M. . Cinnéide, eds.), vol. 6956 of *Lecture Notes in Computer Science*, pp. 127–141, Berlin, Heidelberg: Springer-Verlag, Sep. 2011.

[77] V. Yannibelli and A. Amandi, "A knowledge-based evolutionary assistant to software development project scheduling," *Expert Systems with Applications*, vol. 38, pp. 8403–8413, Jul. 2011.

[78] W.-N. Chen and J. Zhang, "Ant colony optimization for software project scheduling and staffing with an event-based scheduler," *IEEE Transactions on Software Engineering*, vol. 39, pp. 1–17, Jan. 2013.

[79] J. Xiao, X.-T. Ao, and Y. Tang, "Solving software project scheduling problems with ant colony optimization," *Computers and Operations Research*, vol. 40, pp. 33–46, Jan. 2013.

[80] C. Amrit, "Coordination in software development: The problem of task allocation," in *Proceedings of the 2005 Workshop on Human and Social factors of Software Engineering (HSSE 2005)* (G.-C. Roman, W. G. Griswold, and B. Nuseibeh, eds.), pp. 1–7, ACM, May 15-21 2005.

[81] M. Hapke, A. Jaszkiewicz, and R. Slowinski, "Fuzzy project scheduling system for software development," *Fuzzy Sets and Systems*, vol. 67, pp. 101–117, Oct. 1994.

[82] D. A. Callegari and R. M. Bastos, "A multi-criteria resource selection method for software projects using fuzzy logic," in *Enterprise Information Systems* (J. Filipe and J. Cordeiro, eds.), vol. 24 of *Lecture Notes in Business Inforamtion Processing*, pp. 376–388, Berlin, Germany: Springer-Verlag, May 6-10 2009.

[83] C. Stylianou and A. S. Andreou, "Intelligent software project scheduling and team staffing with genetic algorithms," in *Artificial Intelligence Applications and Innovations* (L. S. Iliadis, I. Maglogiannis, and H. Papadopoulos, eds.), vol. 364 of *IFIP Advances in Information and Communication Technology*, pp. 169–178, Berlin, Heidelberg: Springer-Verlag, Sep. 2011.

[84] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[85] S. Gerasimou, C. Stylianou, and A. S. Andreou, "An investigation of optimal project scheduling and team staffing in software development using particle swarm optimization," in *Proceedings of the 14th International Conference on Enterprise Information Systems (ICEIS 2012)* (L. A. Maciaszek, A. Cuzzocrea, and J. Cordeiro, eds.), vol. 2, pp. 168–171, SciTePress, Jun.–Jul. 2012.

[86] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, pp. 4104–4108, IEEE Computer Society, Oct. 1997.

[87] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[88] C. Stylianou and A. S. Andreou, "A multi-objective genetic algorithm for intelligent software project scheduling and team staffing," *Intelligent Decision Technologies*, vol. 7, pp. 59–80, Jan. 2013.

[89] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, Apr. 2002.

[90] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb, "A simulated annealing-based multiobjective optimization algorithm: AMOSA," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 3, pp. 269–283, 2008.

[91] D. A. V. Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithm research: A history and analysis," Tech. Rep. TR-98-03, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, USA, Oct. 1998.

[92] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[93] C. Stylianou and A. S. Andreou, "Investigating the impact of developer productivity, task interdependence type and communication overhead in a multi-objective optimization approach for software project planning," *Advances in Engineering Software*, vol. 98, pp. 79–96, Aug. 2016.

[94] L. L. Minku, D. Sudholt, and X. Yao, "Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis," *IEEE Transactions on Software Engineering*, vol. 40, pp. 83–102, Oct. 2013.

[95] L. L. Minku, D. Sudholt, and X. Yao, "Evolutionary algorithms for the project scheduling problem: Runtime analysis and improved design," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pp. 1221–1228, ACM, Jul. 2012.

[96] V. Yannibelli and A. Amandi, "A memetic approach to project scheduling that maximizes the effectiveness of the human resources assigned to project activities," in *Hybrid Artificial Intelligent Systems* (E. Corchado, V. Snášel, A. Abraham, M. Woźniak, M. Graña, and S.-B. Cho, eds.), vol. 7208 of *Lecture Notes in Computer Science*, pp. 159–173, Berlin, Heidelberg: Springer-Verlag, Mar. 2012.

[97] V. Yannibelli and A. Amandi, "A diversity-adaptive hybrid evolutionary algorithm to solve a project scheduling problem," in *Intelligent Data Engineering and Automated Learning – IDEAL 2014*, vol. 8669, pp. 412–423, Berlin, Heidelberg: Springer-Verlag, Sep. 2014.

[98] V. Yannibelli and A. Amandi, "Project scheduling: A multi-objective evolutionary algorithm that optimizes the effectiveness of human resources and the project makespan," *Engineering Optimization*, vol. 45, pp. 45–65, Apr. 2013.

[99] V. Yannibelli and A. Amandi, "Hybridizing a multi-objective simulated annealing algorithm with a multi-objective evolutionary algorithm to solve a multi-objective project scheduling problem," *Expert Systems with Applications*, vol. 40, pp. 2421–2434, Jun. 2013.

[100] I. D. Steiner, *Group Processes and Productivity*. New York, NY, USA: Academic Press, 1972.

[101] M. di Penta, M. Harman, G. Antoniol, and F. Qureshi, "The effect of communication overhead on software maintenance project staffing: a search-based approach," in *2007 IEEE International Conference on Software Maintenance (ICSM 2007)*, pp. 315–324, IEEE Computer Society, Oct. 2007.

[102] P. M. Institute, *PMI Lexicon of Project Management Terms*. Philadelphia, PA, USA: PMI Publications, 2015.

[103] M. V. Mäntylä and J. Itkonen, "More testers – the effect of crowd size and time restriction in software testing," *Information and Software Technology*, vol. 55, pp. 986–1003, 6 2013.

[104] F. P. Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering*. Boston, MA, USA: Addison-Wesley, anniversary ed., 1995.

[105] T. Abdel-Hamid and S. E. Madnick, *Software Project Dynamics: An Integrated Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, 1991.

[106] M. J. Douglas, *The impacts of the handoffs on software development: a cost estimation model*. PhD thesis, Department of Information Systems and Decision Sciences, College of Business Administration, University of South Florida, Tampa, FL, USA, May 2006.

[107] D. E. Goldberg and R. Lingle Jr, "Alleles loci and the traveling salesman problem," in *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 154–159, L. Erlbaum Associates, Jul. 1985.

[108] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," Tech. Rep. TIK-Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland, May 2001.

[109] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the Pareto archived evolution strategy," *Journal of Evolutionary Computing*, vol. 8, pp. 149–172, Jun. 2000.

[110] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "Design issues in a multiobjective cellular genetic algorithm," in *Evolutionary Multi-Criterion Optimization* (S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, eds.), vol. 4403 of *Lecture Notes in Computer Science*, pp. 126–140, Berlin Heidelberg: Springer-Verlag, Mar. 2007.

[111] "Standard occupational classification," 2010. Available at http://www.bls.gov/soc/classification.htm.

[112] "O*net online," 1998. Available at http://www.onetonline.org/.

[113] F. Luna, D. L. González-Álvarez, F. Chicano, and M. A. Vega-Rodríguez, "The software project scheduling problem: A scalability analysis of multi-objective metaheuristics," *Applied Soft Computing*, vol. 15, pp. 136–148, Feb. 2014.

[114] E. Alba, G. Luque, and F. Luna, "Parallel metaheuristics for workforce planning," *Journal of Mathematical Modelling and Algorithms*, vol. 6, pp. 509–528, Sep. 2007.

[115] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, pp. 760–771, Oct. 2011.

[116] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 257–271, Nov. 1999.

[117] A. M. Cox, "I am never lonely: A brief history of employee personality testing," *Stay Free!*, pp. 1–5, Sep.–Nov. 2003.

[118] F. W. Taylor, *The Principles of Scientific Management*. New York, NY, USA: Harper and Brothers, 1911.

[119] H. Münsterberg, *Psychology and Industrial Efficiency*. Cambridge, MA, USA: The Riverside Press, 1913.

[120] I. B. Weiner and R. L. Greene, *Handbook of Personality Assessment*. Hoboken, NJ, USA: John Wiley and Sons, 1st ed., 2008.

[121] H. C. Link, *Employment Psychology: The Application of Scientific Methods to the Selection, Training and Rating of Employees*. New York City, NY, USA: The Macmillan Company, 1919.

[122] I. B. Myers, M. H. McCaulley, N. L. Quenk, and A. L. Hammer, *MBTI®Manual: A Guide to the Development and the Use of the Myers-Briggs Type Indicator®*. Consulting Psychologists Press, Mountain View, CA, USA, 3rd ed., 1998.

[123] C. G. Jung, *Psychological Types*. London, UK: Routledge and Kegan Paul, 1923.

[124] D. Keirsey and M. Bates, *Please Understand Me: Character and Temperament Types*. Del Mar, CA, USA: Prometheus Nemesis Book Company, 1984.

[125] P. T. Costa Jr and R. R. McCrae, *NEO Inventories Professional Manual*. Psychological Assessment Resources, Odessa, FL, USA, 1992.

[126] E. C. Tupes and R. E. Christal, "Recurrent personality factors based on trait ratings," Tech. Rep. ASD-TR-61-97, Personnel Laboratory, Aeronautical Systems Division, Air Force Systems Command, United States Air Force, Lackland Air Force Base, TX, USA, May 1961.

[127] "Employment testing overview," 2006. Available at http://www.siop.org/Workplace/employment

[128] J. E. Moore, "Personality characteristics of information systems professionals," in *Proceedings of the 1991 ACM SIGCPR Conference on Computer Personnel Research (SIGCPR 1991)* (T. W. Ferratt, ed.), pp. 140–155, ACM, Apr. 1991.

[129] R. B. Cattell, A. K. Cattell, and H. E. P. Cattell, *16PF Select Manual*. Institute for Personality and Ability Testing, Champaign, IL, USA, 1993.

[130] R. B. Cattell, *The description and measurement of personality*. New York, NY, USA: Harcourt, Brace, and World, 1946.

[131] J. L. Wynekoop and D. B. Walz, "Investigating traits of top performing software developers," *Information Technology and People*, vol. 13, pp. 186–195, Sep. 2000.

[132] H. G. Gough and A. B. Heilbrun Jr, *The Adjective Checklist Manual*. Palo Alto, CA, USA: Consulting Psychologists Press, 1983.

[133] D. C. Smith, "The personality of the systems analysts: An investigation," *ACM SIGCPR Computer Personnel*, vol. 12, pp. 12–14, Dec. 1989.

[134] L. F. Capretz, "Personality types in software engineering," *International Journal of Human-Computer Studies*, vol. 58, pp. 207–214, Feb. 2003.

[135] M. A. G. Peeters, H. F. J. M. van Tuijl, C. G. Rutte, and I. M. M. J. Reymen, "Personality and team performance: A meta-analysis," *European Journal of Personality*, vol. 20, pp. 377–396, May 2006.

[136] L. F. Capretz and F. Ahmed, "Why do we need personality diversity in software engineering?," *ACM SIGSOFT Software Engineering Notes*, vol. 35, pp. 1–11, Mar. 2010.

[137] J. S. Karn and A. J. Cowling, "A study of the effect of disruptions on the performance of software engineering teams," in *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE 2005)*, pp. 417–425, IEEE Computer Society, Nov. 2005.

[138] L. Fernández-Sanz and S. Misra, "Influence of human factors in software quality and productivity," in *Computational Science and Its Applications – ICCSA 2011* (B. Murgante, O. Gervasi, A. Iglesias, D. Taniar, and B. Apduhan, eds.), vol. 6786 of *Lecture Notes in Computer Science*, pp. 257–269, Berlin, Heidelberg: Springer-Verlag, Jun. 2011.

[139] S. T. Acuña, M. Gómez, and N. Juristo, "How do personality, team processes and task characteristics relate to job satisfaction and software quality?," *Information and Software Technology*, vol. 51, pp. 627–639, Mar. 2009.

[140] G. A. Dafoulas and L. A. Macaulay, "Facilitating group formation and role allocation in software engineering groups," in *Proceedings of the ACS/IEEE 2001 International Conference on Computer Systems and Applications (AICCSA 2001)*, pp. 352–359, IEEE Computer Society, Jun. 2001.

[141] S. T. Acuña and N. Juristo, "Assigning people to roles in software projects," *Software - Practice and Experience*, vol. 34, pp. 675–696, Jun. 2004.

[142] S. T. Acuña, N. Juristo, and A. M. Moreno, "Emphasizing human capabilities in software development," *IEEE Software*, vol. 23, pp. 94–101, Mar.–Apr. 2006.

[143] M. André, M. G. Baldoquín, and S. T. Acuña, "Formal model for assigning human resources to teams in software projects," *Information and Software Technology*, vol. 53, pp. 259–275, Mar. 2011.

[144] L. F. Capretz and F. Ahmed, "Making sense of software development and personality types," *IT Professional*, vol. 12, pp. 6–13, Jan.–Feb. 2010.

[145] R. H. Rutherfoord, "Using personality inventories to form teams for class projects – a case study," in *Proceedings of the 7th Annual ACM Conference on Information Technology Education (SIGITE 2006)*, pp. 9–14, ACM, Oct. 2006.

[146] V. Pieterse, D. G. Kourie, and I. P. Sonnekus, "Software engineering team diversity and performance," in *Research for a Changing World* (J. Bishop and D. Kourie, eds.), ACM International Conference Proceeding Series, pp. 180–186, Pretoria, South Africa: SAICSIT, Oct. 9-11 2006.

[147] A. R. Peslak, "The impact of personality on information technology team projects," in *Proceedings of the 2006 ACM SIGMIS Conference on Computer Personnel Research (SIGMIS CPR 2006)* (K. Kaiser and T. Ryan, eds.), pp. 273–279, ACM, Apr. 2006.

[148] P. Sfetsos, I. Stamelos, L. Angelis, and I. Deligiannis, "Investigating the impact of personality types on communication and collaboration-viability in pair programming – an empirical study," in *Extreme Programming and Agile Processes in Software Engineering* (P. Abrahamsson, M. Marchesi, and G. Succi, eds.), vol. 4044 of *Lecture Notes in Computer Science*, pp. 43–52, Berlin, Germany: Springer-Verlag, Jun. 2006.

[149] K. S. Choi, F. P. Deek, and I. Im, "Exploring the underlying aspects of pair programming: The impact of personality," *Information and Software Technology*, vol. 50, pp. 1114–1126, Oct. 2008.

[150] J. S. Karn, S. Syed-Abdullah, A. J. Cowling, and M. Holcombe, "A study into the effects of personality type and methodology on cohesion in software engineering teams," *Behaviour and Information Technology*, vol. 26, pp. 99–111, Mar. 2007.

[151] N. Salleh, E. Mendes, J. Grundy, and G. S. J. Burch, "An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, vol. 2, pp. 577–586, ACM, May 2010.

[152] C. Stylianou and A. S. Andreou, "A multi-objective genetic algorithm for software development team staffing based on personality types," in *Artificial Intelligence Applications and Innovations* (L. Iliadis, I. Maglogiannis, and H. Papadopoulos, eds.), vol. 381 of *IFIP Advances in Information and Communication Technology*, pp. 37–47, Berling, Heidelberg: Springer-Verlag, Sep. 2012.

[153] C. Stylianou, S. Gerasimou, and A. S. Andreou, "A novel prototype tool for intelligent software project scheduling and staffing enhanced with personality factors," in *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012)*, vol. 2, pp. 277–284, IEEE Computer Society, Nov. 2012.

[154] B. Kosko, "Fuzzy cognitive maps," *International Journal of Man-Machine Studies*, vol. 24, pp. 65–75, Jan. 1986.

[155] A. S. Andreou and H. C. Neophytou, "Forming effective software development teams using fuzzy cognitive maps," in *Proceedings of the Joint 4th Scientific Conference on Project Management and 1st IPMA/MedNet Conference on Project Management Advances, Training and Certification in the Mediterranean*, pp. 358–363, May 29–31 2008.

[156] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS 1995)*, pp. 39–43, IEEE Computer Society, Oct. 1995.

Appendix A

Optimization Algorithms

This appendix chapter presents the various optimization algorithms that were adopted in the approaches described in Chapters 4–7 and 9. Specifically, the algorithms are presented as they were proposed by their corresponding authors.

A.1 Genetic algorithm

The basic optimization procedure of the genetic algorithm (GA) proposed by Holland [84] is shown in Algorithm A.1.

Algorithm A.1 Genetic algorithm

Set $x_{min}, x_{max}, t = 0, p_c, p_m$
for each individual $i \in POP(t)$ **do**
    $x_i =$ rand(x_{min}, x_{max})
while maximum iterations reached or stopping criteria satisfied **do**
    for each individual $i \in POP(t)$ **do**
        Evaluate the position, x_i, of the individual using objective function $F(x_i)$
    $POP'(t) =$ select$(POP(t))$
    $POP'(t) =$ crossover$(POP'(t), p_c)$
    $POP'(t) =$ mutate$(POP'(t), p_m)$
    Set $POP(t+1) = POP'(t)$
    Set $t = t + 1$

A.2 Particle swarm optimization algorithm

The steps of the particle swarm optimization algorithm (PSO) given in Eberhart and Kennedy [156] are shown in Algorithm A.2.

Algorithm A.2 Particle swarm optimization

Set $x_{min}, x_{max}, v_{max}, t = 0$
for each particle $p_i \in SWARM(t)$ **do**
    $x_i =$ rand(x_{min}, x_{max})
    $v_i =$ rand$(-v_{max}/3, v_{max}/3)$
while maximum iterations reached or stopping criteria satisfied **do**
    for each particle $p_i \in SWARM(t)$ **do**
        Evaluate the position, x_i, of the particle using objective function $F(x_i)$
        if $F(x_i) < F(pbest_i)$ **then**
            $pbest_i = x_i$
        if $F(x_i) < F(gbest)$ **then**
            $gbest = pbest_i$
    for each particle $p_i \in SWARM$ **do**
        $v_i = v_i + (U(0, c_1) \times (pbest_i - x_i)) + (U(0, c_2) \times (gbest - x_i))$
        $x_i = x_i + v_i$
    Set $t = t + 1$

A.3 Nondominated sorting genetic algorithm

The procedure of the nondominated sorting genetic algorithm (NSGA-II), proposed by Deb et al. [89], is given in Algorithm A.3.

Algorithm A.3 Nondominated sorting genetic algorithm (NSGA-II)

$R_t = P_t \cup Q_t$
$F =$ fast-non-dominated-sort(R_t)
where $F = (F_1, F2, \dots)$
$P_{t+1} = \emptyset$ and $i = 1$
until $|P_{t+1}| + |F_i| \leq N$
    crowding-distance-assignment(F_i)
    $P_{t+1} = P_{t+1} \cup F_i$
    $i = i + 1$
Sort$(F_i, \prec_n)$
$P_{t+1} = P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
$Q_{t+1} =$ make-new-pop(P_{t+1})
$t = t + 1$

A.4 Archived multiobjective simulated annealing algorithm

Algorithm A.4 presents the steps of the archived multiobjective simulated annealing algorithm (AMOSA) introduced by Bandyopadhyay et al. [90].

Algorithm A.4 Archived multiobjective simulated annealing (AMOSA)

Set $Tmax, Tmin, HL, SL, \gamma, iter, \alpha, temp = Tmax$
Initialize the $Archive$ with $\gamma \times SL$ solutions
$current\text{-}pt =$ random($Archive$)
while $temp > Tmin$ **do**
    for $i = 0$ **to** $iter$ **do**
        $new\text{-}pt =$ perturb($current\text{-}pt$)
        Check the domination status of $new\text{-}pt$ and $current\text{-}pt$
        if $current\text{-}pt$ dominates $new\text{-}pt$ **then**
            Set $new\text{-}pt$ as $current\text{-}pt$ with probability $prob = \frac{1}{1+exp(\Delta dom_{avg} \times temp)}$
            where $\Delta dom_{avg} = \frac{(\sum_{i=1}^{k} \Delta dom_{i,new\text{-}pt}) + \Delta dom_{current\text{-}pt,new\text{-}pt}}{k+1}$ and k equals the total number of points in the $Archive$ that dominate $new\text{-}pt$, $k \geq 0$
        else if $current\text{-}pt$ and $new\text{-}pt$ are nondominating with respect to each other **then**
            Check the domination status of $new\text{-}pt$ and points in the $Archive$
            if $new\text{-}pt$ is dominated by k $(k \geq 1)$ points in the $Archive$ **then**
                Set $new\text{-}pt$ as $current\text{-}pt$ with probability $prob = \frac{1}{1+exp(\Delta dom_{avg} \times temp)}$
                where $\Delta dom_{avg} = \frac{\sum_{i=1}^{k} \Delta dom_{i,new\text{-}pt}}{k}$
            else if new-pt is nondominating with respect to all the points in the $Archive$ **then**
                Set $new\text{-}pt$ as $current\text{-}pt$ and add $new\text{-}pt$ to the $Archive$
                if $Archive\text{-}size > SL$ **then** Cluster $Archive$ to HL number of clusters
            else if $new\text{-}pt$ dominates k $(k \geq 1)$ points in the $Archive$ **then**
                Set $new\text{-}pt$ as $current\text{-}pt$ and add it to the $Archive$
                Remove all the k dominated points from the $Archive$
        else if $new\text{-}pt$ dominates $current\text{-}pt$ **then**
            Check the domination status of $new\text{-}pt$ and points in the $Archive$
            if $new\text{-}pt$ is dominated by $k (k \geq 1)$ points in the $Archive$ **then**
                Set point of the $Archive$ that corresponds to Δdom_{min} as $current\text{-}pt$ with probability $prob = \frac{1}{1+exp(-\Delta dom_{min})}$ otherwise set $new\text{-}pt$ as $current\text{-}pt$
                where $\Delta dom_{min} =$ minimum of the difference of domination amounts between the $new\text{-}pt$ and the k points
            else if $new\text{-}pt$ is nondominating with respect to the points in the $Archive$ **then**
                Set $new\text{-}pt$ as the $current\text{-}pt$ and add it to the $Archive$
                if $current\text{-}pt$ is in the $Archive$ **then** Remove $current\text{-}pt$ from the $Archive$
                else if $Archive\text{-}size > SL$ **then** Cluster $Archive$ to HL number of clusters
            else if $new\text{-}pt$ dominates k other points in the $Archive$ **then**
                Set $new\text{-}pt$ as $current\text{-}pt$ and add it to the $Archive$
                Remove all the k dominated points from the $Archive$
    $temp = \alpha \times temp$
if $Archive\text{-}size > SL$ **then** Cluster $Archive$ to HL number of clusters

A.5 Multiobjective cellular genetic algorithm

The procedure of the multiobjective cellular (MOCell) genetic algorithm put forward by Nebro et al. [110] is shown in Algorithm A.5.

Algorithm A.5 Multiobjective cellular genetic algorithm (MOCELL)

```
1:  proc Steps_Up(mocell)
2:  Pareto_front = Create_Front();
3:  while not TerminationCondition() do
4:      for individual ←1 to mocell.popSize do do
5:          n_list ←Get_Neighbourhood(mocell,position(individual));
6:          parents ←Selection(n_list);
7:          offspring ←Recombination(mocell.Pc,parents);
8:          offspring ←Mutation(mocell.Pm,offspring);
9:          Evaluate_Fitness(offspring);
10:         Replacement(position(individual),offspring,mocell,aux_pop);
11:         Insert_Pareto_Front(offspring);
12:     mocell.pop ←aux_pop;
13:     mocell.pop ←Feedback(mocell,Pareto_front);
14: end_proc Steps_Up;
```

A.6 Pareto archived evolution strategy algorithm

The steps of the Pareto archived evolution strategy (PAES) algorithm are shown in Algorithm A.6. The algorithm was originally proposed by Knowles and Corne [109].

Algorithm A.6 Pareto archived evolution strategy algorithm (PAES)

1: generate initial random solution c and add it to the archive
2: mutate c to produce m and evaluate m
3: **if** (c dominates m) **then** discard m
4: **else if** (m dominates c) **then**
5: replace c with m, and add m to the archive
6: **else if** (m is dominated by any member of the archive) **then** discard m
7: **else** apply test$(c, m, archive)$ to determine which becomes the new current solution and whether to add m to the archive
8: until a termination criterion has been reached, return to line 2

A.7 Strength Pareto evolutionary algorithm

Algorithm A.7 provides the procedure of the strength Pareto evolutionary algorithm (SPEA) designed by Zitzler et al. [108].

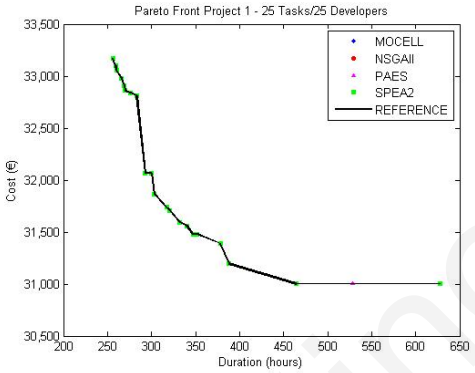
Algorithm A.7 Archived multiobjective simulated annealing (AMOSA)

1: Input: N (*population size*), $\overline{N}$ (*archive size*), T (*maximum number of generations*)
2: Output: A (*nondominated set*)
3: Step 1: ***Initialization***: *Generate an initial population* $\boldsymbol{P}_0$ *and create the empty archive (external set)* $\overline{\boldsymbol{P}}_0 = \emptyset$. *Set* $t = 0$.
4: Step 2: ***Fitness assignment***: *Calculate fitness values of individuals in* $\boldsymbol{P}_t$ *and* $\overline{\boldsymbol{P}}_t$.
5: Step 3: ***Environmental selection***: *Copy all nondominated individuals in* $\boldsymbol{P}_t$ *and* $\overline{\boldsymbol{P}}_t$ *to* $\overline{\boldsymbol{P}}_{t+1}$. *If size of* $\overline{\boldsymbol{P}}_{t+1}$ *exceeds* $\overline{N}$ *then reduce* $\overline{\boldsymbol{P}}_{t+1}$ *by means of the truncation operator, otherwise if size of* $\overline{\boldsymbol{P}}_{t+1}$ *is less than* $\overline{N}$ *then fill* $\overline{\boldsymbol{P}}_{t+1}$ *with dominated individuals in* $\boldsymbol{P}_t$ *and* $\overline{\boldsymbol{P}}_t$.
6: Step 4: ***Termination***: *If* $t \geq T$ *or another stopping criterion is satisfied then set* $\boldsymbol{A}$ *to the set of decision vectors represented by the nondominated individuals in* $\overline{\boldsymbol{P}}_{t+1}$. *Stop.*
7: Step 5: ***Mating selection***: *Perform binary tournament selection with replacement on* $\overline{\boldsymbol{P}}_{t+1}$ *in order to fill the mating pool.*
8: Step 6: ***Variation***: *Apply recombination and mutation operators to the mating pool and set* $\boldsymbol{P}_{t+1}$ *to the resulting population. Increment generation counter* ($t = t + 1$) *and go to Step 2.*

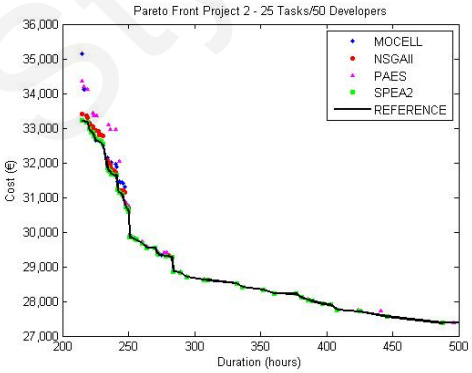
Appendix B

Additional Figures

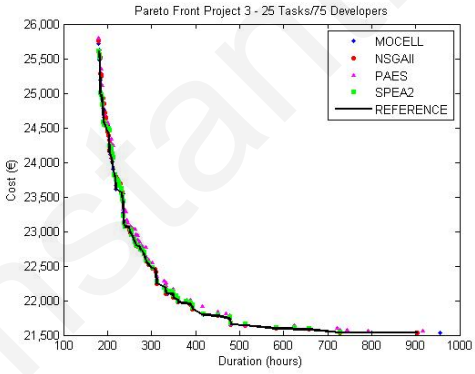
This appendix provides the approximated Pareto fronts for MOCell, NSGA-II, PAES and SPEA2, as well as the reference Pareto front that were generated for the 16 project instances in dataset $DS1$ for the first experiment of Approach 4.



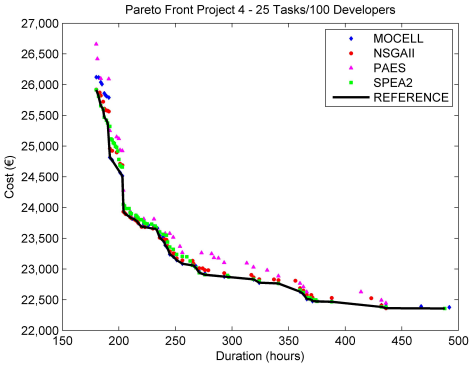
(a) Project $P4A1$



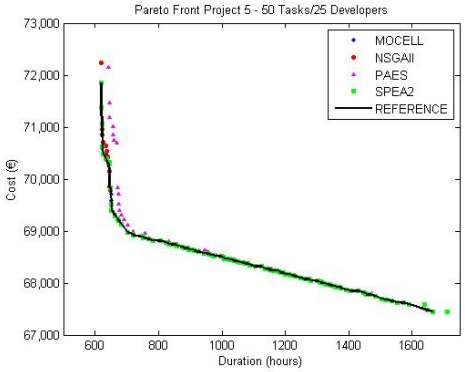
(b) Project $P4A2$



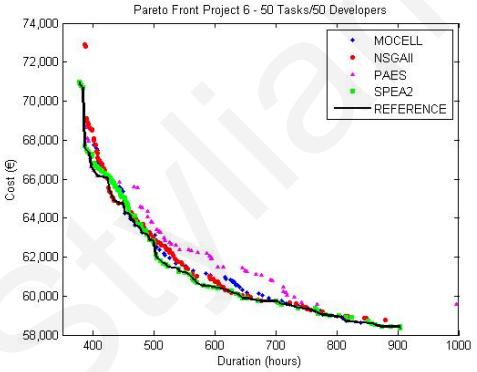
(c) Project $P4A3$



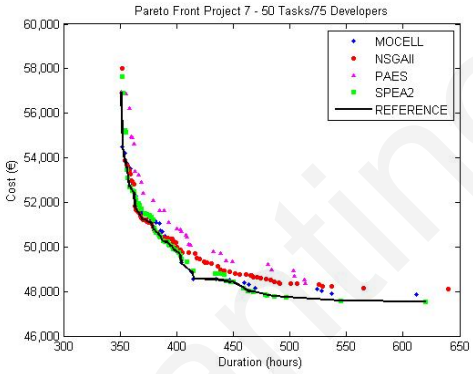
(d) Project $P4A4$



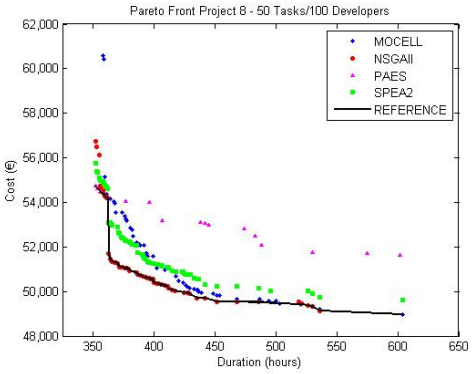
(e) Project $P4A5$



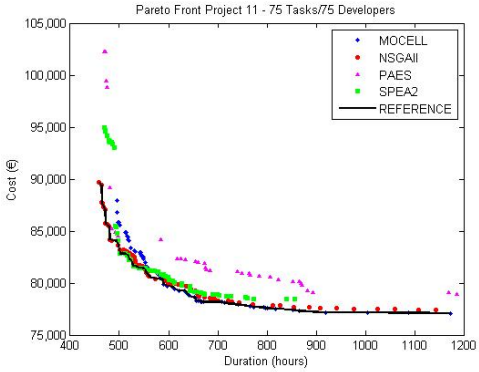
(f) Project $P4A6$



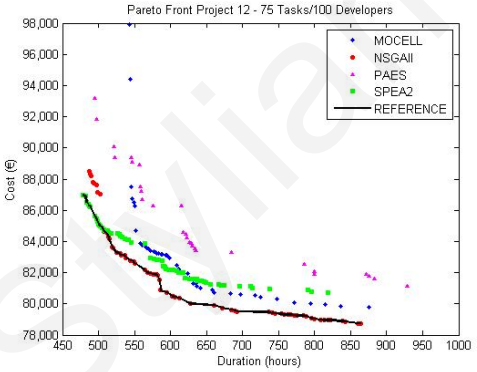
(g) Project $P4A7$



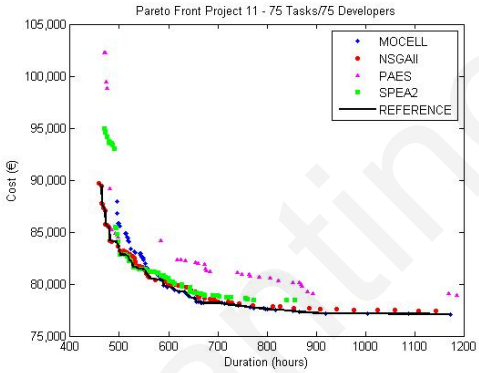
(h) Project $P4A8$



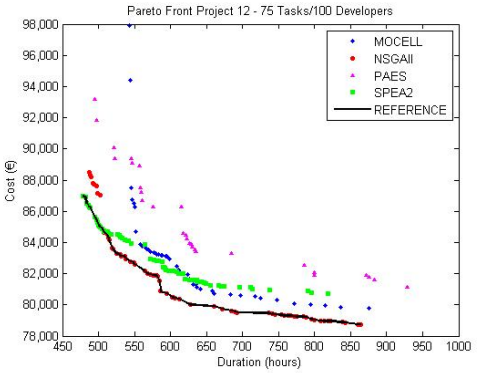
(i) Project $P4A9$



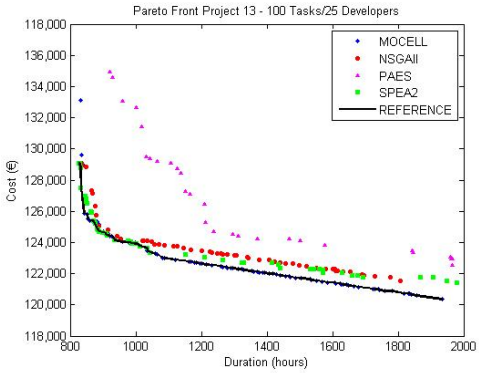
(j) Project $P4A10$



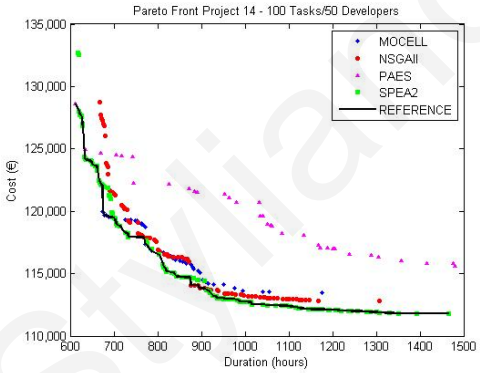
(k) Project $P4A11$



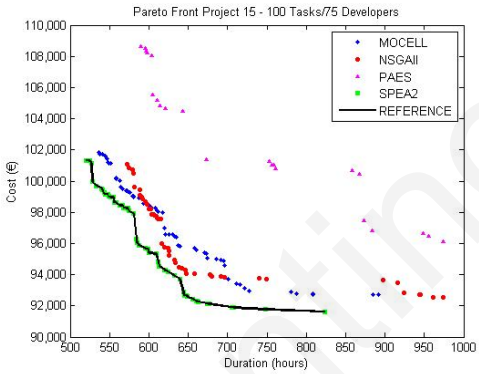
(l) Project $P4A12$



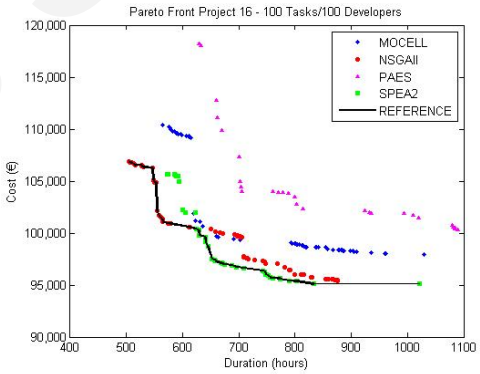
(m) Project $P4A13$



(n) Project $P4A14$



(o) Project $P4A15$



(p) Project $P4A16$

Figure B.1: Pareto fronts corresponding to the best solutions generated by the four variations for the 16 project instances in $DS1$ in Approach 4

Appendix C

Description of Notations

Table C.1 presents the notations used throughout the dissertation regarding the description of the resource-constrained project scheduling problem for software development.

Table C.1: Description of notations

Notation	Description
m	Number of tasks required for the software project
$T = \{t_1, t_2, \ldots, t_m\}$	Set of m tasks required for the software project
D	Set of dependency relationships where $(t_i, t_j) \in D$ if task t_j depends on task t_i
p	Number of skills required by the software project
$S = \{s_1, s_2, \ldots, s_p\}$	Set of p skills required by the software project
$TREQ = [treq_{ik}]$	$m \times p$ logical matrix where $treq_{ik} = 1$ if task t_i requires skill s_k or $treq_{ik} = 0$ if task t_i does not require skill s_k
n	Number of developers available in the software development company
$R = \{r_1, r_2, \ldots, r_n\}$	Set of n developers available in the software development company
$LEXP = [lexp_{jk}]$	$n \times p$ real matrix where $lexp_{jk}$ denotes the level of experience that developer r_j possesses in skill s_k in the range $[0, 1]$
q	Number of software development professions required by the software project
$H = \{h_1, h_2, \ldots, h_q\}$	Set of q software development professions required by the software project
$PROD = [prod_{jl}]$	$n \times q$ real matrix where $prod_{jl}$ denotes the productivity rate of developer r_j in profession h_l in the range $[0, 2]$
t_i^{assigned}	Number of developers assigned to task t_i

Table C.1: Description of notations (continued from previous page)

Notation	Description
$B^i = \{b_1^i, b_2^i, \ldots, b_n^i\}$	Bitset representing the assignment of developers for task t_i
$A^i = \{r_j \mid \forall r_j \in R \wedge b_j^i = 1\}$	Set of t_i^{assigned} developers assigned to task t_i
t_i^{sched}	Scheduled start time of task t_i
t_i^{start}	Actual start time of task t_i
t_i^{finish}	Actual finish time of task t_i
t_i^{duration}	Duration of task t_i
t_i^{skills}	Number of skills required by task t_i
t_i^{prof}	Profession to which task t_i belongs
t_i^{effort}	Effort required for task t_i
t_i^{paths}	Number of communication paths between developers in task t_i
t_i^{overhead}	Communication overhead incurred in task t_i
$Y = \{additive, disjunctive, conjuctive\}$	Set of possible task interdependence types
t_i^{type}	Interdependence type of task t_i taken from the set Y
t_i^{prod}	Overall team productivity rate of developers assigned to task t_i
r_j^{salary}	Salary of developer r_j
$r_j^{\text{pers}} = \{r_j^{\text{N}}, r_j^{\text{E}}, r_j^{\text{O}}, r_j^{\text{A}}, r_j^{\text{C}}\}$	Personality type of developer r_j based on the Five-Factor Model
r_j^{N}	Level of neuroticism of developer r_j
r_j^{E}	Level of extraversion of developer r_j
r_j^{O}	Level of openness to experience of developer r_j
r_j^{A}	Level of agreeableness of developer r_j
r_j^{C}	Level of conscientiousness of developer r_j
$h_l^{\text{pers}} = \{h_l^{\text{N}}, h_l^{\text{E}}, h_l^{\text{O}}, h_l^{\text{A}}, h_l^{\text{C}}\}$	Desired personality of profession h_l based on the Five-Factor Model
h_l^{N}	Level of neuroticism desired for profession h_l
h_l^{E}	Level of extraversion desired for profession h_l
h_l^{O}	Level of openness to experience desired for profession h_l
h_l^{A}	Level of agreeableness desired for profession h_l
h_l^{C}	Level of conscientiousness desired for profession h_l