Using SBSE for Project Management
Optimisation: Finding Robust Project Plans

Stefan Gueorguiev
stefan.gueorguiev@kcl.ac.uk

MSc in Advanced Software Engineering 2007/2008
School of Physical Sciences and Engineering
King’s College London

Supervisor
Mark Harman

September 2008
Abstract

This paper provides an innovative approach to automating risk management for project planning. This is achieved using a search-based approach. The results provide insights on problematic areas in projects. The optimisation algorithms used provide the project manager with a decision support in the preliminary stages of constructing a project plan. The results allow the manager to act proactively to possible pitfalls that may occur during the project execution.
## Future Work

7.1 Algorithms ........................................... 25  
7.2 Extending Robustness ................................. 25  
7.3 Extending the Overall Model ......................... 26  
   7.3.1 Communication Overhead ....................... 26  
   7.3.2 Periodic Restaffing ............................. 26  
   7.3.3 Skill Quality .................................. 26  

## Conclusion  

8  

A Complete Source Code 28  

Bibliography and references 29
List of Figures

2.1 Genetic Algorithm Structure ................................................. 4
3.1 Pareto Front ................................................................. 9
3.2 Sample Result ............................................................... 10
4.1 Queue System ............................................................... 12
4.2 Ordering Genome ............................................................ 13
6.1 Random Search Comparison : Plan A ....................................... 17
6.2 Random Search Comparison : Plan B ....................................... 18
6.3 Random Search Comparison : Database .................................... 18
6.4 Random Search Comparison : SmartPrice .................................. 18
6.5 Random Search Comparison : QuoteToOrder ................................ 19
6.6 Plan B Pareto Front .......................................................... 19
6.7 Plan A Pareto Front .......................................................... 20
6.8 Database Pareto Front ....................................................... 21
6.9 SmartPrice Comparison ..................................................... 21
6.10 Plan A Comparison .......................................................... 22
6.11 QuoteToOrder Comparison ................................................ 22
6.12 SmartPrice with removed dependencies ................................... 23
6.13 SmartPrice and error level comparison .................................... 24
List of Tables

5.1 Table of available projects. ................................................. 14
Chapter 1

Introduction

1.1 Background

Project management is an important part of any software project, even so as of recent as more and more software companies focus on the MDA development style. Project management is the act of defining different tasks within a project and assigning them to particular people. This way a road map is constructed before the project is executed. This road map gives an estimate on the project deadline as well as serves as guidelines on how it is to be executed. Even though project management is crucial to the success of a software project it is one process that has not been automated. This is why, as of recently, it has been a hot topic of research for software engineers. Many of the interesting problems in this field are usually hard for people to do.

One particularly hard problem is the ability to construct a robust project plan and to manage the risks that might come up. In every commercial project there are unforeseen factors and issues that occur as the project is executed. In extreme cases, when deadlines are not met or cost becomes too big, these can potentially lead to the death of the project. What sets good project managers apart is their ability to anticipate such pitfalls and devise a project plan from day one that can accommodate for them. This, however is a difficult task as in a commercial world there are countless external factors that can influence the project execution. One obvious example is when employees leave the company. It is virtually impossible for a project manager to anticipate every single problem that might occur. In reality, to battle this problem, project managers construct their plans in such a way that they would accommodate for possible pitfalls. For example, this can be done by setting certain additional time aside for tackling problems. This approach, however, can lead to unfeasible plan as far as the business side is concerned.

It would be useful for project managers if there existed a set of tools that can aid them in their risk management analysis. Generally, it would be useful to be able to quickly analyse a project definition and produce a report on what the possible project plans are and what are their trade-offs in terms of robustness. This way we can answer questions such as: How much will my deadline be extended if I were to construct a plan that can accommodate for some of pitfalls? What about for large number of pitfalls? or If I am willing to extend my deadline by X number of days, how much more confident will I be about my plan’s ability to accommodate for unforeseen issues?. It is the aim of this project to develop and provide a fully automated set of tools that answer just these questions. Our approach analyses project definition and provides solutions that differ in their robustness. This set of tools can serve the project manager for risk assessment while constructing their plan so they can be better prepared for pitfalls. In essence, we are automating an proactive approach to project planning.

This solution is accomplished with the help of Search-Based Software Engineering. Given a project definition (a set of tasks, staff, dependencies between tasks, etc.) we will be searching the space of all possible project plans. We will be looking for the ones that meet our criteria in minimising the impact of possible pitfalls on projects.
1.2 Road Map

Chapter 2 covers the related literature. This is a list of papers that either served directly as a stepping stone upon which we build on top, or provided food for thought and are related in a more distant way. Chapter 3 provides a description of the problem input, the strategy we have taken to come up with the solution, and finally a discussion on the solution itself. Next, chapter 4 covers in-depth the implementation of our solution strategy. Chapter 5 presents explicitly the research questions that we have attempted to answer, as well as the experimental setup used in obtaining these answers. Chapter 6 gives answers to the research questions via discussions based on the obtained experimental results. Chapter 8 serves as a way-point on what could be done to extend this project. Finally, chapter 9 concludes.
Chapter 2

Literature Review

This chapter will focus on all the work that has been done in this field. We will present in detail relevant work that was used as a stepping stone for this project. That is, work upon which we have built to achieve our goal.

2.1 Search-Based Software Engineering and Genetic Algorithms

Search-Based Software Engineering (SBSE) is a relatively new approach to problem solving when dealing with computer problems. There are many traditional ways to find the best solution. All these attempt to solve the problem by computing the solution systematically. In contrast, SBSE implores techniques to search the space of all possible solutions for one that is good enough. A particular way of achieving this are Genetic algorithms.

2.1.1 SBSE

SBSE is more of an approach to problem solving than an algorithm. In fact all the algorithms that implement this ideology share similarities. They all approach the problem by searching the solution space for the best possible solution, instead of trying to actually solve the problem. There is a large number of different approaches that fall in the category of SBSE. They all share several common traits:

- Problems – Due to the power of SBSE and its computationally expensive nature these approaches are applied mostly to the hardest of problems. These are problems that are at least NP-Complete, where an efficient algorithm to find the adequate solution simply does not exist. This is where SBSE really shines and outperforms its counterparts.

- Optimal Solution – It is often the case that SBSE does not find the most optimal solution. Usually SBSE algorithms converge to a local optimum solution that may or may not be very close the global optimal one. This largely depends on the problem at hand as well as the algorithm used. For certain problems certain algorithms find a solution that is extremely close to the global optimum than others.

- Randomness – SBSE algorithms are usually employing techniques that involve randomness. This means that from one run to the other we may obtain different local optimum solutions. To counter that these algorithms are run a number of times. This ensures that the final solution will be much closer to the local optimum.

- Iterative Approach – Almost all SBSE algorithms start with a randomly generated solution space. Using that, the algorithm performs a number of iteration and in each one it improves the current solution space and thus, the algorithm guides itself toward a optimum solution space.

A more in-detail survey of the state of the art in this area can be found in [6].
2.1.2 Genetic Algorithms

Genetic Algorithms (GAs) are a particular set of algorithms that implement the ideas promoted by SBSE. These algorithms search the solution space for the local optimum solution in an iterative manner. The algorithms are inspired by the Theory of Natural Selection by Charles Darwin. The idea of the algorithm is to simulate an evolution of a population (of solutions) where the solutions that are fit to solve the problem survive and improve, while the ones that are poor at solving it are discarded. The overall structure of a GA is presented in Figure 2.1. Each step is explained in more detail:

- Initialization – The initial population of solutions is generated usually at random. The quality of these is usually very poor.

- Fitness Assignment – Each individual (solution) in the population is assigned a fitness value indicating how good it is at solving the problem at hand.

- Termination Condition – If we have obtained a good enough solution, or we have reached the maximum number of generations (iterations) the algorithm terminates.

- Selection – Out of the population, the most promising/fit solutions are chosen for replication. There are different ways to do this. The most commonly used one is binary tournament selection where two individuals are chosen at random from the population and the one with the better fitness is chosen for replication.

- Recombination – Out of all solutions chosen for replication two are chosen and using a crossover operator they are used to produce two new ones. The idea here is that, because of correctly chosen crossover operator, the new individuals (children) will carry, and often improve, some of the information that made their parents particularly fit to solve this problem.

- Mutation – Some of the solutions are chosen at random and a small mutation is introduced in them. This mutation involves changing the individual slightly in a random manner.
• Reinsertion – The newly created solutions (children) are reinserted in the population. There are different ways to do this. One would be to discard the parent population and completely replace it with the children. Another could be to construct the next population by the fittest children and parents.

They key components of each GA are the genome representation and the fitness function.

• Genome – This is the internal solution representation. More often than not a solution is encoded as an array or a sequence of bits. Each different genome encoding comes with its own mutation and crossover operator. The crossover operator defines how given two individuals we can produce two children. The mutation operator is similar, but instead it changes the current genome by a small amount.

• Fitness Function – This function assigns a numeric value to each genome. The value is usually between 0 and 1 and shows how fit the current individual is to solve our problem, with 0 being the best possible solution and 1 being the worse. Constructing this function correctly is usually one of the hardest parts.

2.2 Multi-Objective Optimisation

Genetic algorithms are in fact solving an optimization problem. They do this by guiding the desired result to minimise or maximise the fitness function. They attempt to search the solution space for one solution that fits a certain criteria best. In our case, however, we are interested in optimising several criterion (we would like to minimise the overall completion time as well as maximise the robustness). To do so we will use an extended version of genetic algorithms. These are a set of algorithms that can find optimal solutions when several optimization criterion exists.

A good introductory tutorial can be found in [5]. There are many ways to accomplish the multi objective optimisation goal. Some include adding weights to the different objectives (making one more important than other) and including this calculation in the fitness function. However this approach has the fault of making assumption that are only known to the end user and cannot be always quantified (e.g. criteria A is three times more important than criteria B). There is solution to this problem. It is a set of multi objective optimisation algorithms such as SPEA2 and NSGA II [9][7]. The key idea of these algorithms is to use a modified fitness function that is able to judge weather a solution is better than another based on all criteria. For simplicity assume that we have \( n \) different criteria that we want to minimise on. Each fitness of a solution is therefore a vector of \( n \) numbers \(< f_1, f_2, ..., f_n >\), with each \( f_i \) (TODO: i between 1 and n) representing the fitness of the solution with respect to the criteria \( i \).

2.3 SBSE in Project Management

Project management is a area full of interesting and hard problems that have not been tackled by software engineers until recently. As such, the research in this field is currently growing rapidly, with new, ground breaking work emerging on a daily basis. We will not attempt to cover everything that has been done here. In fact, entire papers [3] have been written in attempt to summarize all work done in this area. Instead we will look at the work that is relevant to our project. This is the work that we have used as a stepping stone to build upon in our approach.

The largest relevant accomplishment in this area was done by Antoniol et al. [2] described in a lot of detail. These deal with project planning optimisation problems such as to minimise overall completion time. That is, the group answers the important question of “Given a number of tasks and people, what is the best ordering of tasks, assignment of people to teams and finally, assignment of teams to tasks so that the overall project completion time is minimal?” What is more is that this exploration incorporates in its model many real scenarios including:

• Task Dependencies – This models the fact that in every project plan certain tasks depend on the completion of others before they can be accomplished. For example, the task of deploying a beta depends on several others: building a GUI, setting up a database layer and possibly, building a preliminary business logic layer.
• Skill Assignments – Here the simple model is enriched to allow for skill requirements for tasks and skill properties to employees. For example, building the business logic in a .NET project requires the according skill set and within the whole staff there are 5 people who have .NET proficiency while there are 3 people who have HTML proficiency. The last 3 would be more useful when designing the front end of a website.

• Communication Overhead – This incorporates in the model the different communication cost between team members. There are several common communication models and which one is applicable depends on the tasks’ nature. Certain routine tasks (such as maintenance) can require no communication between team members. In that case the time it would take to perform the task is simply equal to the person-month estimation divided by the number of available people. In other cases however, this can be more complicated. With certain tasks that can follow the logarithmic model in which scenario it is not always the case that if we were to add more people to a task it will get completed quicker.

• Periodic Restaffing – It is the practice of project managers to observe the project plan at several milestones and attempt to adjust it if there would be a benefit. This is also incorporated in Antoniol’s model.

Another accomplishment in the above-mentioned papers is the exploration of different SBSE techniques and their efficiency to solve such project management problems. It became clear that GAs are the way to go. They naturally outperformed random search, but also outperformed various different meat heuristic algorithms. In their implementation of the GA two pieces are particularly interesting as we make extensive use of them in our approach. These are the genome choice and the fitness function evaluation.

2.4 Further Readings

An interesting multi-objective approach applied to a similar problem-space is presented by Zhang et al.[4]. In this paper the author presents an approach to the Next-Release problem, where a manager is forced to pick the subset of features of all available as to satisfy as many customers as possible. In the paper it is taken into account several real-world scenarios such as the fact that some customers are more important than others and hence, their requirements for features is more important. The approach uses a novel approach of utilizing multi-objective GAs to achieve its goal. In addition, it presents a comparative study of several multi-objective algorithms (SPEA2, NSGA-II, two-archive) in the context of this problem.
Chapter 3

Problem Statement

In this chapter we would like to present the basic ideas of our solution: the input that defines the problem as well as the overall strategy to achieving the final solution. In addition, we will look at the significance of our solution and how it can be used.

3.1 Project Definition Model

When talking about project plan robustness we are essentially referring to a property of the plan. Given a project definition \((DEF)\) that consists of

- A set \(WPS = \{wp_1, wp_2, ..., wp_n\}\) – set of tasks to be performed.
- A set \(P = \{p_1, p_2, ..., p_m\}\) – set of available resources/staff.
- A set \(DEPS = \{(wp_i, wp_j) : 0 \leq i \leq n \text{ and } 0 \leq j \leq n \text{ and } j \neq n\}\) – set of dependencies between tasks, where \(wp_j\) requires \(wp_i\) to be completed first.
- A set \(S = \{s_1, s_2, s_3, ..., s_n\}\) – set of different skills. Each member of the staff \((P)\) has a skill associated with it and each task in \(WPS\) requires a certain skill to be performed. For example a task called Design Web Front-end may require an HTML skill that only 3 people in the staff of 10 may have.

We are looking to construct a project plan (ordering of tasks in the sequence in which they should be completed without violating dependency constraints as well as assignment of people to teams). Naturally, for most cases there will be many different ways in which it is possible to arrange the given input. Let us assume that we have two possible arrangements (plans) - A and B. One way to differentiate between the two is to look at their overall completion time. Depending on the arrangements of the plan execution A and B will have different overall completion times. This can hint as to which plan is better and the topic was deeply explored by Antoniol et al. [2]. Assume that we have a function \(COMPL(X)\) (described in detail in the previous section) that takes as an argument the project plan and calculates its overall completion time in person-hours. Let us assume that \(COMPL(A) \neq COMPL(B)\).

3.2 Project Robustness a.k.a. Risk Management

3.2.1 Reactive Approach

We are focusing on another quantitative property that the two project plans share, namely robustness. Assume that during the project execution something unforeseen happens (as it usually does in most commercial projects). One option to accommodate for the sudden changes is for the project manager to re-adjust the plan (modify existing teams, re-order tasks, etc.) If this is done with both plans A and B it will result in two new plans - \(A_{\text{post}} - m\) and \(B_{\text{post}} - m\). Usually, due to the change, the overall completion time will change. It could be the case that the new completion time can go over the deadline, which is undesirable. In some extreme cases in the real world the
unforeseen issues are so large that the new project plan goes over the deadline by a huge amount (months or years). This can be so extreme as to force the cancellation of the project due to bigger than expected costs and inability to meet crucial deadlines.

To measure how fit a plan is for unforeseen changes we can simply take the difference \(|\text{COMPL}(A) - \text{COMPL}(A_{post} - m)|\). Lets denote this by \(R_A\) and \(|\text{COMPL}(A) - \text{COMPL}(A_{post} - m)|\) by \(R_B\). The smaller these number are, the more robust the plan. Smaller number means that unexpected changes will affect the overall completion time by the least amount. This way we can conclude that if \(R_A > R_B\) plan B is more robust than A. The reader should keep in mind that plans A and B are different plans for the same project definition.

### 3.2.2 Proactive Approach

So far we described the most usual reaction of project managers (PMs) to problems arising during the project execution. However, this is not the most efficient way to handle issues. What sets good PMs apart is their ability anticipate for possible problems and adjust the project plan beforehand as to accommodate for such pitfalls. An example of a way to do this is to assign fewer resources to a task. This way if a new task emerges half-way through the execution the free resources can be assigned to it and the overall deadline will not change as much. Naturally, such pro-active approach can guarantee better robustness at the expense of larger completion time. However if this time is available and given that the problematic areas have been correctly identified and estimated, the PM can be more certain with their completion time estimation. In our solution we are automating this exact pro-active approach to project planning.

Given a project definition DEF with the aid of multi-objective optimisation (the algorithm is discussed in more detail in the next chapter) we construct a number of project plans. Our two high-level optimisation objectives are the overall completion time and the robustness. In this case, since it is assumed that the project plan is as optimal as possible when it is first constructed no changes are necessary when something goes wrong. The belief holds that the project plan was constructed in such way from the beginning that it will accommodate for any pitfalls as smoothly as possible. Let the original plan be called A and the one where something went wrong (new task was added, etc.) be \(A_{pre} - m\). In this case the robustness \(R_A\) of the plan is simply computed by \(v\text{COMPL}(A) - \text{COMPL}(A_{pre} - m)\). Informally, the robustness of the plan is the completion time difference between the case where nothing actually went wrong and the case where problems arose. Ideally, in spaced out plans with a lot of precautions built into them this number should be 0.

### 3.2.3 Robustness Definition

Before we continue we should formalise what does it mean for “something to go wrong” in the scope of this project. Naturally, there are many real world scenarios that can occur to impact a project execution. The number is far too large and some of these are way too complex to be included in this project. The possible pitfalls include:

- **Task Inflation/Deflation** – It is often the case that the person-month estimates for tasks are not accurate. Therefore as the project progresses these get readjusted. Failing to plan for this can cause large delays, idle staff due to task dependencies, etc.

- **Task Addition/Removal** – During the course of a project it often turns out that new features need to be developed or some necessary components were overlooked. To adjust for that new tasks get added to the plan in arbitrary positions with new dependencies. Depending on their importance and size these can affect the project’s deadline by a large amount.

- **Dependencies** – Task dependencies shape up the project plan execution to a great extend. The number of dependencies can shape up the possible different solutions as well as have a large impact on the robustness of the solutions.

Naturally, in a real world scenario there are many more things that could wrong. For a complete list please refer to the “Future Work” section. Due to the fact that, to the best of our knowledge, this is the first work that explores this problem area and the time constraints we had we have
not included them. However, by the end of this document, it should be clear to the reader that the approach and framework we have built can easily be extended to accommodate for additional pitfall scenarios.

3.3 End Solution

3.3.1 Solution Model

Due to the nature of pro-active project planning we are generating a number of local-optimal solutions, neither of which is better than the other. This is the case because when developing a plan from a project definition based on our overall available time and in-depth knowledge of the project mechanics we can construct a plan with very good robustness, one that is able to accommodate for most possible pitfalls. Naturally, such plan will be very spread out, with lots of gaps and precautions built into it so it will have the largest completion time. On the other hand, if we feel fairly confident that nothing will go wrong we can construct a very tightly packed project plan. This is a plan with extremely poor robustness but very short completion time. There is a clear trade off between risk mitigation and completion time.

Both of these borderline plans, as well as the ones between them, are feasible. In fact all the optimal solutions generate a Pareto front as shown on Figure 3.1. Neither of these solutions is better than the other, as far as solution generation is concerned. It requires a more in-depth knowledge of the company, the nature of the project and many additional factors that cannot be included in a computer model. This knowledge is possessed only by the user – the PM. Therefore, it is our intent to generate all optimal points on the Pareto front and present them to the PM to aid their analysis.

3.3.2 Solution Interpretation

The solution will merely serve as an point of reference to the PM. It will help them to better identify points of interest when considering robustness vs. completion time trade-offs. Looking at the Pareto front, where each point represents different project plan several conclusions can be drawn. If we take a look at the example (Figure 3.1) the graph offers a few conclusions:

- If the slope is very high steep between points A and B. This means that if the PM can afford it, they can switch from plan A to plan B by extending the deadline by a small amount but gaining a large amount of robustness (which translates to lowering the completion time difference). Since both points A and B represent actual plans the PM can also observe the differences in these plans and make conclusion as to what change offers the high increase in robustness.
• When between points C and D the slope is very smooth. This means that for a small increase of robustness the PM will have to pay a large price of completion time difference. Depending on the actual business circumstances this could be desirable or not.

• There exist boundary points. If the deadline is as tight as possible then the PM can observe plan A and possibly take it. If, however, robustness is of the biggest importance and completion time is no problem the PM can easily pick plan X which is constructed in such way that no matter what happens it will always complete by the given deadline. The reader should note that constructing such plans is a very rare scenario.
Chapter 4

The Search-Based Approach

Chapter 3 described the strategy we have taken to solving this problem. Here we will discuss in more detail the implementation. Results accomplished in similar problems [2] have shown that GAs outperform other search-based meta heuristics such as Hill Climbing and Simulated Annealing. This is why we have also opted to use a GA approach.

The overall implementation closely follows what is presented by Di Penta et al. in [1]. This includes the overall problem model, the genome and fitness assignment via simulation. Because we use make heavy use of all this we will briefly describe it here. For more details please refer to above-mentioned paper.

4.1 Problem Model

All WPs form a set of non-overlapping tasks with possible dependency constraints between them. Each task may also have a skill requirement which means that the task can be completed only by a set of people that have the needed skill. In addition, all available staff is distributed into teams consisting of one or more people. Every person has a skill as well. It should be noted that people with different skills can be in the same team. However when this team is assigned to a WP only the people with the required skill can do work for the task while the rest stay idle. This is an impractical solution and, indeed, such solutions are quickly discarded by our search algorithm as they produce poor times.

Our final solution is an ordering of tasks (in terms of precedence for completion) without violating dependency constraints, as well as an assignment of people to teams and then – teams to tasks. This helps to model the result as a queuing system where each customer (WP) is a element in the queue and waits its turn to be eventually assigned to a server (a team). When a WP is completed it leaves the system. When the queue is empty then all tasks have been completed and the project is considered finished. We should note that different solutions (e.g. different ordering of WPs) represent different project plans for the same project definition.

It should be noted that for this paper we have disregarded Brook’s law. That means that the amount of time to complete a task is \( t_{req} \), where \( t_{req} \) is the estimated person-hours required to complete the task and \( n \) is the number of people in the team that is assigned to it.

4.2 Multi-Objective Approach

In our approach we have a number of objectives to optimise. These are:

- \( O_1 \) – Overall completion time.
- \( O_2 \) – Completion time difference (robustness) when new tasks get added or removed. The amount of tasks to be added is pre-defined by a error level - X%. This means that for inflation we have picked X% of the tasks at random and inflated their required time by a randomly chosen amount that ranges from 1 person-hour up to double their original size. It should be noted all the random choices are generated every time when we evaluate the objective. This means that at every evaluation newly chosen random tasks will get inflated. Finally, when
the evaluation is complete all the changes are discarded from the original plan. This prevents from gradually modifying the plan itself.

- $O_3$ – Completion time difference when tasks get inflated or deflated in terms of required person-hours. This is again governed by an error level – $Y\%$. In the case of adding WPs, we randomly create $Y\%$ more tasks. We, then, pick a random duration ranging from the minimum duration of all the existing tasks to their overall maximum duration. Finally, we insert the newly created task at a random position in the queue. As with evaluating $O_2$ all random choices are generated every time an evaluation is needed.

Therefore each solution has an associated with it an optimisation vector of size 3 : $<O_1, O_2, O_3>$. Due to the definitions of each objective we will always seek to minimise each vector. Therefore solutions with smaller vectors correspond to fitter solutions that will yield closer-to optimal results. To achieve this we have used multi-objective GAs.

4.3 Simulation Technique

The fitness value of each genome is constructed from the completion time of the project it represents. To calculate this we have used a staffing simulation shown in Figure 4.1 (courtesy of Di Penta et al.[1]). All tasks are seeded into a queue and are assigned to teams on a first come, first serve basis. More information can be found in the paper mentioned above, which has developed this method.

4.4 The Genome

As described in previous chapters the genome is one of the important parts of a GA. It is essentially the internal problem representation. In our case we are using a compound Ordering Genome consisting of two arrays shown in Figure 4.2. The first one represents the WP ordering in the incoming queue while the second one represents the assignment of people to teams. The crossover and mutation operator for the staffing array are fairly simple. For crossover we use a single-point crossover and the mutation operator simply picks at random two positions in the array and exchanges their teams.

The WP array is more complicated. The mutation operator is to pick two WPs and exchange their position in the queue. The crossover operator is a variation of the single-point crossover, however it was amended to make sure that there are no collisions, as two WPs cannot be in the same position in the queue. The operator picks at random a position in the array - $i$. It then copies all elements from 0 to $i$ in the first child and it crosses them out from the second parent. Next, it copies all the remaining elements from the second parent in the remaining cells in the first child. To construct the second child, the operator picks the first $n - i$ elements (where $n$ is the size of the array) from the second parent and it copies them in the second child, while removing them from the first parent. Finally, it copies all the remaining elements from the first parent into the second child. This way we ensure that there will be no repetitions in both children. For discussion on possible pitfalls with these choices as well as their resolution please refer to [1].
Figure 4.2: Ordering Genome
Chapter 5

Experimental Setup

The objective of this study is to provide grounds for future development in proactive risk management analysis for project management. We intend to pinpoint the trade-offs between creating a quick project plan and creating a safe (in terms of meeting the deadline) one. The context of our study consists of five real-world project plans that were implemented in the industry recently. Statistics for each project can be found in attached table. To put things into perspective we will provide some background on the nature of the projects:

- **Project A** – A massive maintenance project concerned with fixing the Y2K problem in a large financial software system from a European financial organisation. All tasks in this project are routine ones and require no special skill or any order of completion.

- **Project B** – Aimed to deliver the next release of a large data-intensive, multi-platform software system, written in several languages, including DB II, SQL and .NET.

- **QuoteToOrder** – Medium sized project implemented in a large Canadian sales company. This project aims to add a new enhancement to the supply chain of the company by allowing instant and on-demand conversion of quotes to orders. This change is both internal and customer facing and ultimately affects every level of the organisation (web, internal development, database, sales reps, customers). Most of the employees were involved eventually in training sessions.

- **SmartPrice** – A customer-facing enhancement to the sales process of the same sales organisation. This feature provided more adequate pricing mechanism as well as a method for discounts, voucher use, pricing conversion, etc. The enhancement influences directly all of the revenue stream of the company and as such, extensive QA was involved. This feature affected mostly the web portion of the company’s infrastructure with smaller impact on the underlying database and other internal software. The project ended with adequate employee training to ensure that the features were used properly by everyone.

- **Database** – A massive database upgrade migrating old but crucial Oracle-forms based system to the newest version of Oracle. The information that was migrated was worth millions of dollars and was the basis of the organisation’s operations. About half of the project involves taking numerous precautions against possible causes for data loss. This project involved

<table>
<thead>
<tr>
<th></th>
<th>Project A</th>
<th>Project B</th>
<th>QuoteToOrder</th>
<th>SmartPrice</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Work Hours</td>
<td>4,287</td>
<td>594</td>
<td>544</td>
<td>1,569</td>
<td>5,390</td>
</tr>
<tr>
<td>WPs</td>
<td>84</td>
<td>120</td>
<td>64</td>
<td>79</td>
<td>115</td>
</tr>
<tr>
<td>Staff</td>
<td>20</td>
<td>20</td>
<td>9</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>Different Skills</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Has Dependencies</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.1: Table of available projects.
mainly the DBA section of the company, however development was also briefly involved in the end for training and upgrading the existing scripts and triggers to make use of the newly available functionality.

5.1 Research Questions

The research questions of this study aim to investigate the following:

5.1.1 RQ1: Is our approach better than random search?

This is merely a sanity check. Since, to the best of our knowledge, this is the first attempt to solve such a problem, we need to check if our algorithm is better than the worst possible one. It should be noted that critical path analysis is merely a slightly more sophisticated version of random search. To obtain similar results with critical path analysis one would have to evaluate all possible combinations of project plans given a project definition.

5.1.2 RQ2: Does the obtained Pareto fronts give us any important information?

If the Pareto front constituted of our solutions is smooth, or even worse – linear, then we have obtained the expected result with simple to see trade off between completion time and robustness. However, if there are changes of slopes we will be able to comment on them and draw conclusions.

5.1.3 RQ3: Is there any difference between adding/removing tasks or simply inflating existing ones?

At first glance inflating a certain task may seem as a particular case of adding a new tasks. It seems equivalent to picking a task and adding a new one right after it with the same skill requirements. However this may not always be the case because a separate task allows a possibly idle team to be assigned to it, whereas inflating a task means that the same team will have to service all of it.

5.1.4 RQ4: Is there a relation between the amount of dependencies and robustness?

Intuitively, when a project has fewer dependencies then there are more possible ways in which we can order the tasks for completion priority. This guarantees a larger result size and more possible error-free arrangements. We will explore if this is always the case.

5.1.5 RQ5: What is the impact of different error levels on the project robustness?

There is a big difference between knowing that 10% of the tasks can get inflated and knowing that this can happen to 60% of them. We will explore how the different magnitudes of error affect the robustness.

5.2 Empirical Study Settings and Analysis Method

This section enumerates all the parameters that we have chosen for our experiments. The multi-objective GA parameters are as follows:

- SPEA-II with elitism of 10 was used as a multi-objective GA.
- Population composed of 250 individuals.
- 180 generations for Project A and B, 250 generations for the rest.
- Mutation probability - 0.1, crossover probability - 0.8.
• To reduce randomness each experiment was repeated 30 times and the Pareto front was constructed from all obtained solutions.

All research questions, except RQ5, were performed with error level of 30%. This means that for inflation we have picked 30% of the tasks for modification. In the case of adding WPs, we have randomly created 30% more tasks.

It should be noted that due to the nature of heuristics the obtained solution is by no means the most optimal one. Instead of obtaining global optimum we will obtain the local one. We hope that with the use of numerous runs we have gotten as close to the optimal solution as possible.
Chapter 6

Experimental Results and Analysis

This section addresses each research question, one at a time and attempts to answer it to the best of our knowledge. As mentioned before we have a 3-objective optimisation algorithm. Therefore the Pareto fronts are usually in 3D. For simplicity we have split each Pareto front in two. The overall completion time (objective one) always remains on the X axis. For the Y axis we alternate completion time difference when adding/removing tasks and when inflating/deflating tasks. That is, the Y axis always represents robustness (lower values represent higher robustness), just our interpretation changes. It is important to note that this separation was done after the results were generated.

6.1 RQ1: GA vs Random Search Comparison

Figures 6.1 to 6.5 show the comparison between the Pareto fronts produced by our GA algorithm versus a random search implementation. Given that we want to minimise both quantities it is obvious that our algorithm produces much better Pareto front. This was expected and merely serves as a sanity check that establishes our credibility.

6.2 RQ2: Reading the Pareto fronts

Next, let us look at the Pareto front itself. It is interesting to see the amount of information that we can infer from it. For all following results the Y axis (completion time difference a.k.a robustness) is taken to be the completion time difference when the possible error adding new tasks during the execution. First, consider the Figure 6.6 where we have the result of running our algorithm on Plan B from the mentioned set of plans. Here we can see a regular Pareto front. Each point on it represent a different project plan constructed from the project definition of Plan B. We can clearly

Figure 6.1: Random Search Comparison : Plan A
CHAPTER 6. EXPERIMENTAL RESULTS AND ANALYSIS

Figure 6.2: Random Search Comparison : Plan B

Figure 6.3: Random Search Comparison : Database

Figure 6.4: Random Search Comparison : SmartPrice
CHAPTER 6. EXPERIMENTAL RESULTS AND ANALYSIS

Figure 6.5: Random Search Comparison: QuoteToOrder

Figure 6.6: Plan B Pareto Front
see the trade-offs between picking the different plans. For example if we pick the left-most plan we can be fairly certain that our project can be completed in 35 units of time if things go as expected. However, if during execution of the plan, about 30% more tasks get added we can expect to go over our deadline by 7 units of time (20% extension). On the other hand, if we focus on the other end of the Pareto front and take the right-most plan we can be certain of a few things. That project plan is constructed in such a loose way, with several precautions taken, such that if things go wrong it can accommodate for that. The new tasks can be fit in our loose plan and this will not result in any additional time over the given deadline. Naturally, the price for this is a larger overall completion time. Using this type of analysis and more in-depth knowledge the project manager can be aided by the Pareto front to make decisions. If they are fairly certain that nothing will go wrong, or they are pressed by time they can pick a plan closer to the left-most one. If, however, they expect things to go wrong (because of some additional business-related factors) and would like to keep to the deadline they promise as close as possible they can choose a plan on the right side of the front. One that has low completion time difference (and hence – higher robustness).

Next, let us look at a slightly more complicated result. In Figure 6.7 we can see the results from running our algorithm on Plan A. This Pareto front is far more interesting than the previous one and offer more information. We can see that the first few solutions on the left form a steep slope. This means that the increase in robustness (decrease in completion time difference if things go wrong) is far greater than the increase of overall completion time. The trade off here is in favour of robustness. A PM can pick one plan over the previous knowing that they are gaining large amount of robustness, on average 30%, for just a small increase in total time cost.

After the forth plan though the slope changes drastically and becomes almost a straight line. This segment is highly infavourable since it represents plans where for large increase of completion time gain in robustness is very small. A result of this analysis can be that if the PM is given upto 240 units of time, they can confidently claim that with almost the same certainty they can complete the project in 225 units of time. Another similar segment appears towards the end where the effort to make a flawless plan, one that will not change when the expected amount of pitfalls occur, comes with a large expense of completion time.

Another point of interest that is worth discussing is at the forth plan from left to right. That plan is right at the transition of the slopes. The significance of this is that up to this plan all cheap (with respect to time) optimisations were performed and the next ones are the less desirable and more expensive ones. Keeping in mind that each point represents an actual project plan with a Gantt chart, the PM can view the first four. Observing the differences and optimisations made can provide some useful and perhaps not so obvious information that will allow them to better tune their own project plan. The changes between the first four plans have a big enough impact to be at least reviewed and considered.

Finally, let us look at a result from running our algorithm on the Database project presented on Figure 6.8. Here we can see a different flavour of solutions. The whole front here has a quite steep. This means that all improvements in terms of robustness come at a cheap completion time cost. This is due to the nature of the project. As mentioned before this was a data-sensitive
project that affected the entire infrastructure and revenue stream of the organisation. As such, a lot of precautions were originally taken against something going wrong. And if something were to go wrong, the project was designed so that there were many recovery routes. These include more available staff than necessary among others. In a sense, when the project definition was given, it was already constructed with robustness in mind. This is why it is easy for our algorithm to quickly improve the robustness at low cost. We can see that the slope starts to become horizontal only toward the end, where the final effort to produce a truly robust plan, one that will not change it’s deadline at all if something were to go wrong, is becoming harder and harder.

6.3 RQ3: Inflating existing tasks vs. adding/removing tasks

At first glance the two project alterations may seem very similar. Extending the required time of a task can be viewed as a case of adding a new task with the exact same parameters right after an existing one. However this assumption is not accurate as it fails to take into account several factors. For one, when a task is extended the same team that is assigned to it from the start has to service it until the end. In contrast, a new task can be serviced simultaneously if there is a team available. To show that the relation does not hold let us look at several results. For this section each results consists of two graphs. The left one is when we have taken robustness (Y axis) to be the case when tasks get inflated, whereas the right one is when robustness is the case when new tasks get added. Figure 6.9 shows the two fronts for the SmartPrice project. We can observe the differences of the fronts. When adding new tasks (the right one) the algorithm cannot even get to the point of constructing a truly robust plan. The difference becomes more obvious when we look at the comparison for Plan A (Figure 6.10). Here we can see that the two fronts have a completely
different shape to begin with. All of the interest points (where we can observe a change of slope) or even the different slopes from the left graph do not exist in the right one. In fact, it turns out that for this particular plan it is easier to improve robustness when the problem at hand is the emerging of new and unexpected tasks. The reason behind this, for this particular project, is two-fold. First, the number of available staff compared to the required time for completion is the largest of all possible projects. In addition, the task duration is relatively small. This means that more often there are idle teams not doing anything. When introducing new tasks we can simply reduce the idle time of teams as they can be assigned to them.

Finally, the results of the comparison for the QuoteToOrder project (Figure 6.11) yet again confirm our observations. Here we can see that the Pareto front between the two cases has again taken a completely different shape with each having its own points of interest and corresponding implications.

6.4 RQ4: Effect that dependencies have on robustness

The QuoteToOrder fronts (Figure 6.11) raised a few interesting questions. The project itself is not small enough to produce only 3 points. However, looking at the project definition we noticed a large amount of dependancies. In fact, tasks were laid out in such a way that the whole definition resembled very much a chain of tasks, most laid out to be performed in a consecutive manner. Naturally, the more dependencies there are in the project definition the less is the freedom for the PM (and our algorithm) to construct different plans. To observe the impact that dependencies have on the effort of finding a robust project plan we took a project definition and compared its original Pareto front with the one produced when removing 40% and 80% of its dependencies. The
experiment was performed on the SmartPrice project as its original front provides an adequate number of solutions. The results are shown in Figure 6.12. As expected the number of available plans on the Pareto front decreases by 25% when removing 40% of the dependencies and by additional 17% when removing another 40%. The change of the shape of the front is also expected as with removing the dependencies the project effectively changes its nature and new interest points appear.

What is more interesting is that the need for our algorithm decreases as well. For each scenario we calculated the average of the following ratios – \( r = \frac{\delta_{ct}}{ct} \), where \( \delta_{ct} \) is the Y-axis or the completion time difference if something goes wrong and \( ct \) is the overall completion time – X-axis. This ratio represents the percentage of the overall completion time (X-axis) that we can expect to go over deadline if something were to go wrong. For the case where all dependencies were present this is 37%. This means that, on average, if the project is planned to take 200 days and something goes wrong we can expect to go over our 200 day deadline by 74 days. When we remove 40% of the dependencies this ratio dropped to 25% and finally, when we remove 80% of the dependencies the ratio falls down to 15%. Experimenting with the rest of the plans confirmed this trend. This allows us to make the conclusion that the less dependencies we have the less we can expect to go over our deadline if something were to go wrong. This makes our solution particularly useful for project definitions that rich on dependencies.

6.5 RQ5: Impact that different error levels have on robustness

Finally, let us look at the effect that different error levels have on the robustness of a plan, if any. So far we have assumed that 30% of the things can go wrong. Figure 6.13 shows the study performed on the SmartPrice project with error levels of 10, 30 and 60 percent. The results are intuitive. Since the project remains the same the shape of the front is not altered much. In fact the first interest point (where the slope changes from almost vertical to semi-horizontal) has remained in all fronts. We can see that as we increase the error level the front expands. This is as expected because we simply introduce more work, therefore the deadline that accommodates for this possibility extends and so does the completion time difference if we DO introduce this work.
Figure 6.13: SmartPrice and error level comparison
Chapter 7

Future Work

To the best of our knowledge, this paper is the first that attempts to solve the problem of risk management and mitigation for project management. Therefore it is clear that we were not able to touch every possible aspect and scenario of the problem. In this chapter we will discuss possible future explorations that might provide interesting results.

7.1 Algorithms

The development of multi-objective GAs is constantly evolving. Even now there are a number of algorithms that provide optimal results for different problems. In our approach we have used SPEA-II, however this does not mean that NSGA-II or the newly developed Two-Archive algorithm cannot perform better. One interesting exploration would be to see which algorithm is the best for this kind of problem. Various statistical analysis can be used to show this.

In fact, it may very well be the case that for certain project types one algorithm performs better and for others it is inferior. Or it could be the case that two algorithms find different but equally good or complimenting Pareto fronts. The definition of a Pareto front is such that it allows to easily combine the solutions produced from a number of algorithm in one pool and out of it – a better Pareto front can be constructed.

7.2 Extending Robustness

In our initial model we have included only two possible things that can go wrong during a project execution. As we all know, there are many other factors that can affect a project as it is implemented. Some of these are easier to model than others. Some are nearly impossible because they take into account too many external factors (global economy, etc.) We will try to list all possible pitfalls that might be interesting to add to this study:

- Variable Staffing – The longer the project spans in time, the higher is the chance that some of the staff might simply leave the company. As we know the staff of a company does not remain constant all the time. This is an interesting factor because it cannot be predicted easily, although some knowledge of the employees can help. It would be interesting to add this to our robustness model and to allow the staff to both increase and decrease at any given point of time.

- Dependencies – A common issue that arises during the project execution is the fact that dependencies between tasks arise. Because of hidden or not-so-obvious functionality all of a sudden it turns out that task B requires task A to be completed first. The emerging of new dependencies can change the shape of the project plan greatly. This is why it would be interesting what is the impact on robustness vs. completion time if precautions against such cases are incorporated in the solutions.
7.3 Extending the Overall Model

There are several ways we can extend our problem model as to bring it closer to a real-world scenario.

7.3.1 Communication Overhead

In our approach we have disregarded Brook's Law. However, it is an essential part of most software projects. It is very rarely the case that when we add more people to a team, the required time for the team to complete a task reduces proportionately. One possible way to extend this project is to add support for different communication models and observe their influence on robustness.

7.3.2 Periodic Restaffing

Our approach has automated the pro-active approach to risk management in project planning. Although this approach is generally considered better, it will be interesting to quantify this. To do so, one can implement the reactive approach and then compare the results in terms of robustness. The reactive (opportunist) approach essentially advocates re-staffing and re-ordering of tasks when something goes wrong.

7.3.3 Skill Quality

In the real world two people who are proficient in a skill are rarely equally good. It is often the case that we notice, for example, that person A is a better .NET programmer than person B. We have omitted this in our model but it could add a whole other dimension to the project staffing model.
Chapter 8

Conclusion

This paper presented an innovative approach that automates proactive risk management for project planning, using a search-based solution. Given a project definition, project plans were generated that provide a near-optimal optimisation in terms of robustness. These plans share the property that they accommodate for possible pitfalls that may occur during the project execution in such way that the deadline will be affected by the least possible amount. In addition, we were able to provide a set of plans that optimise on the trade off of robustness versus overall completion time. The proposed models were applied to several real-world projects that have been executed with commercial success over the past 4 years.

Since this is the first solution of this kind we have shown (RQ1) that it performs better than the brute force approach. Our solution produces much more optimal and useful solutions. We have also shown (RQ2) how our solution can be used to aid project managers in their preliminary analysis phase as well as while constructing the plan itself. We have provided results that can be used to pinpoint problematic areas of a project as well as other points of interest that might be worth looking into. When analysing the effects of two different possible problems (RQ3) we have shown that these are not related and should be looked at separately. In addition we have shown (RQ4) that task requirements and dependencies have a large impact on the stability of the project plan. The more dependencies there are, the less stable the plan is. Finally, we have given concrete results (RQ5) behind the reasoning that different error levels cause larger deviations from the deadline.

In conclusion, the paper has demonstrated that multi-objective search-based optimization can be used to provide insights into the project’s stability as well as techniques to manage for pitfalls that can occur during the project’s execution. The paper also provides a first of a kind model framework in this problem area. This framework can be easily further enriched with real-world scenarios that will give us insights on how to reduce a problem that plagues a large amount of software projects currently.
Appendix A

Complete Source Code
Bibliography


