APPLICATIONS OF OPTIMIZATION METHODS IN INDUSTRIAL
MAINTENANCE SCHEDULING AND SOFTWARE TESTING

Kivanc Doganay

2014

School of Innovation, Design and Engineering
Abstract

As the world is getting more and more competitive, efficiency has become a bigger concern than ever for many businesses. Certain efficiency concerns can naturally be expressed as optimization problems, which is a well studied field in the academia. However, optimization algorithms are not as widely employed in industrial practice as they could. There are various reasons for the lack of widespread adoption. For example, it can be difficult or even impossible for non-experts to formulate a detailed mathematical model of the problem. On the other hand, a scientist usually does not have a deep enough understanding of critical business details, and may fail to capture enough details of the real-world phenomenon of concern. While a model at an arbitrary abstraction level is often good enough to demonstrate the optimization approach, ignoring relevant aspects can easily render the solution impractical for the industry. This is an important problem, because applicability concerns hinder the possible gains that can be achieved by using the academic knowledge in industrial practice.

In this thesis, we study the challenges of industrial optimization problems in the form of four case studies at four different companies, in the domains of maintenance schedule optimization and search-based software testing. Working with multiple case studies in different domains allows us to better understand the possible gains and practical challenges in applying optimization methods in an industrial setting. Often there is a need to trade precision for applicability, which is typically very context dependent. Therefore, we compare our results against base values, e.g., results from simpler algorithms or the state of the practice in the given context, where applicable.

Even though we cannot claim that optimization methods are applicable in all situations, our work serves as an empirical evidence for the usability of optimization methods for improvements in different industrial contexts. We hope that our work can encourage the adoption of optimization techniques by more industrial practitioners.
Abstract

As the world is getting more and more competitive, efficiency has become a bigger concern than ever for many businesses. Certain efficiency concerns can naturally be expressed as optimization problems, which is a well studied field in the academia. However, optimization algorithms are not as widely employed in industrial practice as they could. There are various reasons for the lack of widespread adoption. For example, it can be difficult or even impossible for non-experts to formulate a detailed mathematical model of the problem. On the other hand, a scientist usually does not have a deep enough understanding of critical business details, and may fail to capture enough details of the real-world phenomenon of concern. While a model at an arbitrary abstraction level is often good enough to demonstrate the optimization approach, ignoring relevant aspects can easily render the solution impractical for the industry. This is an important problem, because applicability concerns hinder the possible gains that can be achieved by using the academic knowledge in industrial practice.

In this thesis, we study the challenges of industrial optimization problems in the form of four case studies at four different companies, in the domains of maintenance schedule optimization and search-based software testing. Working with multiple case studies in different domains allows us to better understand the possible gains and practical challenges in applying optimization methods in an industrial setting. Often there is a need to trade precision for applicability, which is typically very context dependent. Therefore, we compare our results against base values, e.g., results from simpler algorithms or the state of the practice in the given context, where applicable.

Even though we cannot claim that optimization methods are applicable in all situations, our work serves as an empirical evidence for the usability of optimization methods for improvements in different industrial contexts. We hope that our work can encourage the adoption of optimization techniques by more industrial practitioners.
Acknowledgments

First of all, I would like to thank my supervisors Markus Bohlin, Paul Pettersson, Sigrid Eldh, and also my co-authors in all papers, without whom this thesis would not be possible.

I shall thank the ITS-EASY crowd; both the organizing team, supporters at various forms, and also the other students. Each event has been both instructional and very much fun.

Both MdH and SICS are great workplaces that I very much enjoy being part of. Thanks to my great colleagues, both places have served very valuable experiences to me. I would like to thank everybody at MdH and SICS, for creating a nice and fruitful work environment.

I want to give a special shout-out to my fellows that I have shared many rides while traveling to ATAC project meetings. Even though my probability of sitting in the middle was disproportionately high, it has always been fun to hang out.

Lastly, you the reader; thank you. If you came this far without skipping any lines, you have read at least one page of my thesis. How cool is that! I hope you enjoy the rest as well.

Kivanc Doganay
Västerås, September 2014
Acknowledgments

First of all, I would like to thank my supervisors Markus Bohlin, Paul Pettersson, Sigrid Eldh, and also my co-authors in all papers, without whom this thesis would not be possible.

I shall thank the ITS-EASY crowd; both the organizing team, supporters at various forms, and also the other students. Each event has been both instructional and very much fun.

Both MdH and SICS are great workplaces that I very much enjoy being part of. Thanks to my great colleagues, both places have served very valuable experiences to me. I would like to thank everybody at MdH and SICS, for creating a nice and fruitful work environment.

I want to give a special shout-out to my fellows that I have shared many rides while traveling to ATAC project meetings. Even though my probability of sitting in the middle was disproportionately high, it has always been fun to hang out.

Lastly, you the reader; thank you. If you came this far without skipping any lines, you have read at least one page of my thesis. How cool is that! I hope you enjoy the rest as well.

Kivanc Doganay
Västerås, September 2014
List of Publications

Papers Included in the Licentiate Thesis

1. **Paper A**
   A Tool for Gas Turbine Maintenance Scheduling
   Markus Bohlin, Kivanc Doganay, Per Kreuger, Rebecca Steinert, and Mathias Wärja.
   In proceedings of the Twenty-First Conference on Innovative Applications of Artificial Intelligence (IAAI-09), pages 9-16, Pasadena, California, USA, July 2009.

2. **Paper B**
   Maintenance Plan Optimization for a Train Fleet
   Kivanc Doganay and Markus Bohlin.
   In proceedings of the 12th International Conference on Computer System Design and Operation in Railways and other Transit Systems (COMPRAIL 2010), pages 349-358, Beijing, China, August 2010.

3. **Paper C**
   An Integrated Adaptive Maintenance Concept
   Martin Aronsson, Markus Bohlin, Kivanc Doganay, Anders Holst, Tommy Kjellqvist, and Stefan Östlund.
   In proceedings of the 2010 International Conference on Condition Monitoring and Diagnosis, pages 623-626, Tokyo, Japan, September 2010.

4. **Paper D**
   Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study
   Kivanc Doganay, Markus Bohlin, and Ola Sellin.

1The included articles have been reformatted to comply with the licentiate layout.
List of Publications

Papers Included in the Licentiate Thesis


---

1The included articles have been reformatted to comply with the licentiate layout.


Related Papers not Included in the Licentiate Thesis


Contents

I Thesis 1

1 Introduction 3
   1.1 Background 4
      1.1.1 Maintenance Schedule Optimization 4
      1.1.2 Search-based Software Testing 6
   1.2 Thesis Outline 8

2 Research Summary 9
   2.1 Motivation and Problem Statement 9
   2.2 Research Contributions 10
      2.2.1 Overview of the Papers 12
   2.3 Research Methodology 16

3 Related Work 17
   3.1 Maintenance Schedule Optimization 17
   3.2 Search-based Software Testing 19

4 Conclusions 21
   4.1 Future Work 22

Bibliography 25

II Included Papers 31

5 Paper A: 33
   A Tool for Gas Turbine Maintenance Scheduling 33
      5.1 Introduction 35
## Contents

5.1.1 Related Work .................................. 36  
5.2 Background .................................. 37  
  5.2.1 Improved Analytical Lifetime Predictions .......... 37  
5.3 Problem Description ............................ 38  
  5.3.1 Duration Models ................................ 39  
  5.3.2 Optimization Model ............................ 39  
  5.3.3 Complexity .................................. 42  
5.4 A Tool for Maintenance Scheduling ................. 43  
  5.4.1 MAINTOPT and the Optimization Algorithm ....... 44  
5.5 Development and Deployment ........................ 45  
  5.5.1 First Versions ................................ 46  
  5.5.2 Second Version ............................... 47  
  5.5.3 Deployment at SIT AB ........................... 47  
  5.5.4 Application Maintenance and Support ............. 48  
5.6 Estimated and Measured Benefits .................... 48  
  5.6.1 Results ................................... 49  
  5.6.2 Comparison with CPLEX ........................ 50  
5.7 Conclusions and Future Work ....................... 52  
Bibliography ........................................ 53  

6 Paper B:  
Maintenance Plan Optimization for a Train Fleet 55  
  6.1 Introduction .................................. 57  
    6.1.1 Vehicle maintenance .......................... 57  
  6.2 Related Work .................................. 59  
  6.3 Optimization Model ............................. 60  
  6.4 Evaluation .................................... 63  
    6.4.1 Results ................................... 65  
  6.5 Conclusion and Future Work ....................... 66  
Bibliography ........................................ 67  

7 Paper C:  
An Integrated Adaptive Maintenance Concept 69  
  7.1 Background and Relevance ........................ 71  
  7.2 Vision ........................................ 72  
  7.3 Case Study .................................... 73  
    7.3.1 Brake Pads .................................. 74  
    7.3.2 Condition Monitoring of Pantograph Contact Strip 75  
    7.3.3 Maintenance Planning ......................... 77
<table>
<thead>
<tr>
<th>7.4 Conclusion and Discussion</th>
<th>78</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>81</td>
</tr>
</tbody>
</table>

### 8 Paper D:
Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study

<table>
<thead>
<tr>
<th>8.1 Introduction</th>
<th>85</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1.1 IEC 61131 Control Systems</td>
<td>86</td>
</tr>
<tr>
<td>8.1.2 Related Work</td>
<td>88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8.2 Search-Based Test Case Generation for a POU</th>
<th>88</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.1 MC/DC for Function Blocks Diagrams</td>
<td>89</td>
</tr>
<tr>
<td>8.2.2 Stateful POU's</td>
<td>90</td>
</tr>
<tr>
<td>8.2.3 Coverage Measurement</td>
<td>91</td>
</tr>
<tr>
<td>8.2.4 Hill Climbing</td>
<td>91</td>
</tr>
<tr>
<td>8.2.5 Random Testing</td>
<td>94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8.3 Experimental Evaluation</th>
<th>94</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.1 Experimental Results</td>
<td>95</td>
</tr>
<tr>
<td>8.3.2 Discussion of the Results</td>
<td>98</td>
</tr>
<tr>
<td>8.3.3 Threats to Validity</td>
<td>99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8.4 Conclusions and Future Work</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>101</td>
</tr>
</tbody>
</table>

### 9 Paper E:
MOS: An Integrated Model-Based and Search-Based Testing Tool for Function Block Diagrams

<table>
<thead>
<tr>
<th>9.1 Introduction</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2 Preliminaries</td>
<td>106</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9.3 Tool Overview</th>
<th>107</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3.1 Model-Based Test Generation for FBDs</td>
<td>108</td>
</tr>
<tr>
<td>9.3.2 Search-Based Software Testing for FBDs</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9.4 Case Study</th>
<th>114</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4.1 Results</td>
<td>115</td>
</tr>
<tr>
<td>9.4.2 Implications</td>
<td>116</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9.5 Conclusions</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.6 Acknowledgments</td>
<td>117</td>
</tr>
<tr>
<td>Bibliography</td>
<td>119</td>
</tr>
</tbody>
</table>
10 Paper F:
Search-based Testing for Embedded Telecommunication Software with Complex Input Structures: An Industrial Case Study 121
10.1 Introduction ........................................... 123
10.2 Background ........................................... 124
  10.2.1 Hill Climbing .................................. 124
  10.2.2 (1+1)EA ......................................... 125
  10.2.3 Random Search ................................. 125
10.3 System Under Test .................................... 125
  10.3.1 Analysis and Instrumentation .................. 126
  10.3.2 Execution Environment .......................... 127
10.4 Experimental Evaluation ............................. 128
  10.4.1 Results ......................................... 129
10.5 Discussion ............................................ 133
  10.5.1 Discussion on the Results ...................... 133
  10.5.2 Discussion on the Case Study ................. 133
  10.5.3 Threats to Validity ............................. 134
10.6 Related Work ......................................... 134
10.7 Conclusion and Future Work .......................... 135
10.8 Acknowledgments ..................................... 136
Bibliography ............................................... 137
I

Thesis
Chapter 1
Introduction

Many optimization algorithms that work well in an academic setting do not propagate fully into the industry, for various reasons. For example, it can be difficult to come up with the precise mathematical model of a real-world phenomenon. While an arbitrary abstraction for the model of the research problem is often good enough to demonstrate the optimization approach, it can easily render the solution impractical for the industry.

Another issue is the scalability of a particular optimization approach, typically with respect to the execution time. In an academic study it is often acceptable to use a smaller sized problem instance, in order to demonstrate the basic principals of a particular optimization approach. However, to be used in the industry the approach needs to scale to the industrial sized problems as well. Such practical considerations are not always major considerations for the academic study, but ignoring them may lead to solutions inapplicable in the industrial context.

Oftentimes, the business needs a good enough solution in short enough time, rather than the exact optimum of a model. We can exploit this in certain situations, for example by using heuristic algorithms. However, what is good enough can be subjective or arbitrary. Therefore it is necessary to compare our results against a base value, e.g., results from simpler algorithms or the state of the practice in the given context, where applicable.

We study these challenges of industrial optimization problems, in the form of four case studies at multiple companies in two different domains. The first domain is maintenance scheduling for heavy machinery. Maintenance of gas turbines manufactured and maintained by Siemens Industrial Turbomachinery
Many optimization algorithms that work well in an academic setting do not propagate fully into the industry, for various reasons. For example, it can be difficult to come up with the precise mathematical model of a real-world phenomenon. While an arbitrary abstraction for the model of the research problem is often good enough to demonstrate the optimization approach, it can easily render the solution impractical for the industry.

Another issue is the scalability of a particular optimization approach, typically with respect to the execution time. In an academic study it is often acceptable to use a smaller sized problem instance, in order to demonstrate the basic principals of a particular optimization approach. However, to be used in the industry the approach needs to scale to the industrial sized problems as well. Such practical considerations are not always major considerations for the academic study, but ignoring them may lead to solutions inapplicable in the industrial context.

Oftentimes, the business needs a good enough solution in short enough time, rather than the exact optimum of a model. We can exploit this in certain situations, for example by using heuristic algorithms. However, what is good enough can be subjective or arbitrary. Therefore it is necessary to compare our results against a base value, e.g., results from simpler algorithms or the state of the practice in the given context, where applicable.

We study these challenges of industrial optimization problems, in the form of four case studies at multiple companies in two different domains. The first domain is maintenance scheduling for heavy machinery. Maintenance of gas turbines manufactured and maintained by Siemens Industrial Turbomachinery
AB (SIT AB) is the first case study. Results of this study is deployed in the form of a software program, and is currently used by SIT AB. The second case study in the maintenance scheduling domain is the train fleet maintenance at EuroMaint AB. Maintenance scheduling for train units is further complicated by indirect relations between different units, such as shared costs and workshop resource constraints.

The second domain is software testing for embedded systems. Search-based software testing (SBST) translates testing goals into optimization problems, where metaheuristic search algorithms can be applied. In recent years, SBST methods have been shown to perform well on relatively small scale examples of open source software. Applying SBST on industrial level code is an understudied topic, especially for embedded software. We apply the SBST approach on two different industrial systems; Bombardier Transportation’s train control software and Ericsson’s telecommunications platform. These two case studies pose further challenges due to the implementation paradigms that are encountered in embedded software, and hence are not well investigated in the SBST literature.

1.1 Background

1.1.1 Maintenance Schedule Optimization

Maintenance scheduling is an important problem in application domains that employ sophisticated heavy machinery, such as airplanes or trains, as time spent on maintenance disrupts the normal operation. However, industrial breakdowns can have effects ranging from high costs due to production losses to catastrophic consequences, including physical injury and loss of life. For example, if the pantograph contact strip of a train is not maintained in time this can cause damage to the overhead electricity line and affect operation of other trains on the same line, which would lead to severe operation interruptions and high costs. In order to avoid such breakdowns, maintenance strategies often include preventive maintenance.

A simple and straightforward maintenance strategy is to maintain each component in the system at its predetermined cycle, usually measured in operation time, distance, or similar. As an example, consider a personal car. If the engine oil degrades to an unsafe condition, a warning light on the dashboard would indicate this to the driver. In such a situation the driver should turn off the engine, investigate the problem, and probably change the oil at a nearby mechanic. This
1.1 Background

is analogous to an industrial breakdown, though not necessarily catastrophic or
dramatic. Nevertheless, it disrupts the normal operation and causes extra costs.
Typically, the car manufacturer advises us to change the engine oil regularly,
for example every 8500 km, in order to prevent such breakdown scenarios, and
hence is preventive maintenance.

Moreover, a car would have other components that need maintenance. Let’s
say the air filter should be cleaned at every 8000 km. We can simply co-allocate
all air filter cleaning and oil change maintenance activities, and do them at the
same time to avoid revisiting the repair shop frequently, even though the engine
oil could wait for 500 km more before maintenance. The car manufacturer
would already co-allocate all maintenance activities into a nice cyclic schedule,
also called a block maintenance schedule, so that the car owners do not have to
bear the inconvenience of visiting the maintenance shop too often.

Co-allocating later maintenance items with earlier maintenance items re-
duces the number of maintenance stops, but it increases the actual maintenance.
In the car example, we would be changing the engine oil more often than it
is necessary. This is probably not a big deal for a car owner, as changing the
engine oil is relatively cheap. However, maintenance of heavy machinery is
very costly, and much of it can be avoided.

Condition based maintenance (CBM) is a strategy that aims to reduce the
maintenance cost by doing maintenance based on the actual condition of com-
ponents, instead of using re-determined fixed intervals. Ideally a CBM strategy
would continuously (or frequently) monitor each component and predict when
the component will deteriorate below a safety threshold. The predetermined
maintenance intervals are typically quite conservative and assume worst case
deterioration scenarios. As a component ages, inspecting the component gives a
better estimate about how much of its lifetime has been depleted. CBM simply
leverages this information to do maintenance when it is needed, rather than at
fixed intervals, which should ideally reduce maintenance costs.

However, the industry is often reluctant to adopt CBM or similar dynamic
maintenance strategies. Optimizing a block maintenance schedule is a sin-
gle time affair, which can be done manually or semi-manually. Dynamically
computed component lifetimes (as in CBM) renders such manual work less
practical, as the schedules should be re-optimized when the remaining lifetime
of a component is updated based on new condition data. Therefore, it becomes
more important to have automatic optimization methods.

In our work, we conduct industrial case studies where maintenance schedules
of multi-component systems are optimized. Apart from maintenance stop
related costs, we also model business constraints of each industrial case, such
as parallelism of maintenance activities, available labor at a given time, or spare parts re-used by different units in a fleet. The maintenance schedule optimization problem is further discussed in detail, in Chapter 5–7.

1.1.2 Search-based Software Testing

It is well known that software testing is an expensive activity [1, 2, 3]. Thus considerable research has focused on automating different test activities, notably software test data generation. In recent years, the use of metaheuristic search algorithms have shown promising results in automating parts of software testing efforts. The approach of using metaheuristic search techniques to automatically generate test data is commonly referred to as search-based software testing (SBST).

The fundamental idea of SBST is to translate a testing task into a search problem, such that a generic metaheuristic search algorithm can be applied. A common testing task is finding input data that executes different sections of the system under test, or structural testing. For example, consider the following code segment:

```c
void foo(int x, int y) {
    if (x >= 5)
        if (y >= 10)
            if (x + y < 20)
                // target
        // target
}
```

In order to achieve high branch coverage, we need to find a test inputs that execute the true and false branches of each if statement. This leads to multiple target branches. Let’s focus on a single one, namely the branch that is executed when all if statements are true (marked as target in the above code snippet). The key ingredient of SBST is to translate the goal of executing a particular branch into an objective function. For branch coverage, normalized sum of approach level and branch distance is commonly used as the objective function. Approach level represents how close the execution was to the desired branching node. For example the input vector <x=10,y=5> would diverge from the desired path at the second if statement, which is one step early than the desired one, and therefore gives the approach level of 1.

Branch distance, on the other hand, gives a measure of how far the input vector to make the last if statement true. The same input vector (<x=10,y=5>) would give a branch distance of |5 − 10| = 5, as the value of y is compared to

SBST has been successfully used for various test data generation problems, not only for structural coverage but also other purposes such as functional testing and non-functional testing. The objective function needs to reflect the testing purpose. For instance, consider generating test data for analyzing worst case execution time, where the goal is to find test inputs that take longer to execute the system. Then a reasonable objective function is the execution time itself,
Chapter 1. Introduction

... as parallelism of maintenance activities, available labor at a given time, or spare parts re-used by different units in a fleet. The maintenance schedule optimization problem is further discussed in detail, in Chapter 5–7.

1.1.2 Search-based Software Testing

It is well known that software testing is an expensive activity [1, 2, 3]. Thus considerable research has focused on automating different test activities, notably software test data generation. In recent years, the use of metaheuristic search algorithms have shown promising results in automating parts of software testing efforts. The approach of using metaheuristic search techniques to automatically generate test data is commonly referred to as search-based software testing (SBST).

The fundamental idea of SBST is to translate a testing task into a search problem, such that a generic metaheuristic search algorithm can be applied. A common testing task is finding input data that executes different sections of the system under test, or structural testing. For example, consider the following code segment:

```c
void foo(int x, int y) {
    if (x >= 5) {
        if (y >= 10) {
            if (x+y<20) {
                // target
            }
        }
    }
}
```

In order to achieve high branch coverage, we need to find a test inputs that execute the true and false branches of each `if` statement. This leads to multiple target branches. Let's focus on a single one, namely the branch that is executed when all `if` statements are true (marked as `target` in the above code snippet). The key ingredient of SBST is to translate the goal of executing a particular branch into an objective function. For branch coverage, normalized sum of approach level and branch distance is commonly used as the objective function.

Approach level represents how close the execution was to the desired branching node. For example the input vector `<x=10, y=5>` would diverge from the desired path at the second `if` statement, which is one step early than the desired one, and therefore gives the approach level of 1.

Branch distance, on the other hand, gives a measure of how far the input vector to make the last `if` statement true. The same input vector ( `<x=10, y=5>`) would give a branch distance of `|5 - 10| = 5`, as the value of `y` is compared to 6.

For the example input vector of `<x=10, y=5>`, the combined approach level and normalized branch distance would be `1 + (1 - α⁻⁵) ≈ 1.005`. This value shows how close the input vector to reach the target branch. An input vector that executes the target branch would have the objective value of 0. Hence this formulation of branch coverage problem translates into a minimization problem with an optimum at 0, where various search-based optimization algorithms can be applied. The objective function for a section of possible `<x, y>` values is visualized in Figure 1.1.

SBST has been successfully used for various test data generation problems, not only for structural coverage but also other purposes such as functional testing and non-functional testing. The objective function needs to reflect the testing purpose. For instance, consider generating test data for analyzing worst case execution time, where the goal is to find test inputs that take longer to execute the system. Then a reasonable objective function is the execution time itself,
leading to a maximization problem. Other examples of search-based testing for different purposes are mentioned in the related work section (Section 3.2).

In our work we primarily focused on structural testing, although not only on branch coverage. However the main idea stays the same; the objective function captures the notion of how close we are to cover a certain structure. Further details, such as adapted objective functions and search algorithms, are discussed in individual case studies in Chapter 8–10.

1.2 Thesis Outline

The rest of the thesis is organized as follows: In Chapter 2 we discuss the contributions of the thesis. Chapter 3 presents the related work. Conclusions and future work are discussed in Chapter 4. Afterwards, the technical contributions of the thesis are presented in the form of six research papers, from Chapter 5 to Chapter 10.
Chapter 2

Research Summary

2.1 Motivation and Problem Statement

The two main concerns of our research can be informally summarized as “what can be done?” and “how useful is it?”. In the first domain of maintenance scheduling, the schedules are manually decided by the human operators. Even though experienced operators produce relatively good schedules, due to the high number of directly or indirectly related components finding the best (with respect to costs) schedule becomes a difficult combinatorial problem. Maintenance of business critical machinery typically leads to high monetary costs, not only because of the spare parts but also due to the down time and overhead associated with maintenance stops. Hence, finding schedules that are slightly better (even by a small percentage) than the existing methods can lead to significant monetary gains for the business. However, results need to be applicable in practice to have any tangible benefit for the business. The first research driving question is how the existing data from the business can be transformed into an optimization problem that captures the essential elements of the business operations (RQ1). In our research we also aim to quantify the gain from using an optimization approach (RQ3).

The second domain where we apply search-based optimization algorithms is software testing. Search-based software testing (SBST) has flourished in academia during the last two decades. However, SBST is not yet adapted as a viable option to consider in the software intensive industries. Furthermore, the literature on using search-based testing methods for embedded industrial software is very limited. This leads to the question of applicability of SBST
for such systems (RQ2). Once again, we are also interested in the performance of the search algorithms, but this time with respect to satisfying testing goals (RQ3).

The research aims that are mentioned above are summarized in the form of the following research questions:

RQ1: How can the data from industrial systems be used to optimize operations and maintenance scheduling?

RQ2: How can we apply search-based testing methods for industrial-level embedded systems software?

RQ3: In these two domains, how does optimization methods perform compared to other methods?

2.2 Research Contributions

The contributions of the thesis are presented in the form of research publications. The overall contributions of the licentiate thesis can be summarized as follows.

In Paper A, we explain how maintenance schedules of a particular class of heavy machinery (gas turbines) can be optimized. This serves as an answer to the research question RQ1: “How can the data from industrial systems be used to optimize operations and maintenance scheduling?”. In the paper, the maintenance scheduling problem for gas turbines is studied in detail as an industrial case study, by using the data from Siemens Industrial Turbomachinery AB. As part of the study, we capture the costs and constraints of the maintenance schedules (or the business logic) in a mathematical form, which makes it possible to use generic optimization algorithms. For evaluation, two different optimization algorithms (limited discrepancy search and CPLEX; a mixed integer programming optimization engine) are used and compared against the existing (human crafted) maintenance schedules. This comparison quantifies the benefits of the optimization approach as reduced monetary costs, which relates to the research question RQ3.

Paper B further investigates the maintenance schedule optimization problem in another industrial context; railway vehicles. This time the problem is further complicated with the presence of costs and constraints that relate to the indirect interactions between different units (trains). We construct an optimization model that captures these fleet level costs and constraints, as well as the usual direct maintenance costs and constraints. Once again the CPLEX optimization
2.2 Research Contributions

Engine is used to find better schedules, which again exemplifies how to optimize maintenance schedules (RQ1). A comparison against the cyclic block maintenance schedules (a standard form of maintenance planning practice) is given to quantify the merits of our approach, as an answer to the research question RQ3 in another industrial context.

Paper C positions the maintenance schedule optimization research (that is discussed in paper A and B) in the bigger context of maintenance operations. Low level sensor data acquisition, aggregation of the data, schedule optimization, and high level decision making concerns are addressed in relation to each other as a continuous chain process. This can be seen as a more general answer to the question of how data from industrial systems can be used to optimize operations (RQ1), or a holistic approach towards the goal of better and smarter industrial maintenance.

The next three papers (paper D, E, and F) discuss the use of optimization algorithms in the software testing domain. Paper D is a case study of search-based software testing for an embedded industrial software, namely Bombardier’s train control software. The software under test has certain special properties that are not typically found in SBST studies. For example branch coverage, a common testing goal for SBST, do not make sense as there are no meaningful branches in the given software. Instead we adapt to the structure of the system under test by considering modified condition decision coverage (MC/DC) at variable assignments (instead of branches), which relates to the question of how to use SBST in such systems (RQ2). In the same study we also measure the performance of the SBST techniques, in answer to the research question RQ3.

Paper E is about the same type of system that is used as a case study in paper D, but this time usage of search-based methods in combination and comparison to a model-based approach is discussed. We argue that the combination of the two methods can yield better overall results by complementing each other’s shortcomings, and lead to higher confidence. This paper positions our SBST work in a bigger context of software testing in relation to other testing approaches, and serves as an answer to RQ2 with a wider perspective.

Paper F is another case study, this time on Ericsson’s telecommunications software, again addressing applicability (RQ2) and performance (RQ3) concerns. The system under test has a special execution environment that complicates the practical implementation of the SBST techniques. Another complication arises due to the complex data structures and non-trivial variables such as uninitialized pointers, which are mostly unsupported in the SBST literature. We parse existing test cases to automatically decide on how to initialize dynamic data structures, and also to deduce certain contextual information in order to
Table 2.1: Mapping of research questions to the publications.

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>A, B, C</td>
</tr>
<tr>
<td>RQ2</td>
<td>A, B</td>
</tr>
<tr>
<td>RQ3</td>
<td>D, E, F</td>
</tr>
<tr>
<td>RQ4</td>
<td>D, F</td>
</tr>
</tbody>
</table>

produce practically relevant test data (RQ2).

While answering the research questions we can not claim that our approach and results are applicable to all industrial systems. However, we contribute to the scientific understanding through evidence in the form of case studies. In short, through multiple industrial case studies we demonstrate how optimization algorithms can be applied for the benefit of the industry, and quantify the effects, as an answer to the informal questions of “what can be done?” and “how useful is it?”.

2.2.1 Overview of the Papers

In this section, we give an overview of the papers that are included in the thesis. The contributions of the papers in relation to the research questions are summarized in Table 2.1.

Included Papers


**Abstract:** We describe the implementation and deployment of a software decision support tool for the maintenance planning of gas turbines. The tool is used to plan the maintenance for turbines manufactured and maintained by Siemens Industrial Turbomachinery AB (SIT AB) with the goal to reduce the direct maintenance costs and the often very costly production losses during maintenance downtime. The optimization problem is formally defined, and we argue that feasibility in it is NP-complete. We outline a heuristic algorithm that can quickly solve the problem for
practical purposes, and validate the approach on a real-world scenario based on an oil production facility. We also compare the performance of our algorithm with results from using mixed integer linear programming, and discuss the deployment of the application. The experimental results indicate that downtime reductions up to 65% can be achieved, compared to traditional preventive maintenance. In addition, using our tool is expected to improve availability with up to 1% and reduce the number of planned maintenance days with 12%. Compared to a mixed integer programming approach, our algorithm not optimal, but is orders of magnitude faster and produces results which are useful in practice. Our test results and SIT AB’s estimates based on operational use both indicate that significant savings can be achieved by using our software tool, compared to maintenance plans with fixed intervals.

My contribution: I mainly worked on modeling and implementing the automatic transformation of the scheduling problem into a discrete form that the optimization engine can handle, as well as translating the optimized results back into the problem domain. I also run and documented the experiments that are presented in the paper.


Abstract: Maintenance planning is an important problem for railways, as well as other application domains that employ machinery with expensive replacements and high down-time costs. In a previous paper, we have developed methods for efficiently finding optimized maintenance schedules for a single unit, and proposed that the maintenance plan should be continuously re-optimized based on the condition of components. However, fleet-level resources, such as the availability of expensive spare parts, have largely been ignored. In this paper, we extend our previous approach by proposing a solution for the fleet level maintenance scheduling problem with spare parts optimization. The new solution is based on a mixed integer linear programming formulation of the problem. We demonstrate the merits of our approach by optimizing instances of maintenance schedules based on maintenance data from railway companies operating in Sweden.
My contribution: I was the main author of the paper.


**Abstract:** In this paper, we present a novel maintenance concept based on condition monitoring and dynamic maintenance packaging, by showing how to connect the information flow from low-level sensors to high-level operations and planning under uncertainty. Today, condition-based maintenance systems are focused on data collection and custom-made rule based systems for data analysis. In many cases, the focus is on measuring "everything" without considering how to use the measurements. In addition, the measurements are often noisy and the future is unpredictable which adds a lot of uncertainty. As a consequence, maintenance is often planned in advance and not replanned when new condition data is available. This often reduces the benefits of condition monitoring. The concept is based on the combination of robust, dynamically adapted maintenance optimization and statistical data analysis where the uncertainty is considered. This approach ties together low-level data acquisition and high-level planning and optimization. The concept has been illustrated in a context of rail vehicle maintenance, where measurements of brake pad and pantograph contact strip wear is used to predict the near future condition, and plan the maintenance activities.

My contribution: This paper was written with equal contributions from the authors. I was the corresponding author and the main editor. My main contribution was the sections related to maintenance schedule optimization.


**Abstract:** This paper presents a case study of search-based test generation for embedded system software units developed using the Function Block Di-
2.2 Research Contributions

gra (FBDs), a graphical language in the IEC 61131-3 standard aimed at programmable logic controllers (PLCs). We consider 279 different components from the train control software developed by Bombardier Transportation, a major rail vehicle manufacturer. The software is compiled into C code with a particular structure. We use a modified hill climbing algorithm for generating test data to maximize MC/DC coverage for assignments with logical expressions in the C code, while retaining the semantics of the original FBD implementation. An experimental evaluation for comparing the effectiveness (coverage rate) and the efficiency (required number of executions) of hill climbing algorithm with random testing is presented. The results show that random testing performs well for most units under test, while around 30% of the artifacts significantly benefit from the hill climbing algorithm. Structural properties of the units that affect the performance of hill climbing and random testing are also discussed.

My contribution: I was the main author of the paper.


**Abstract:** In this paper we present a new testing tool for safety critical applications described in Function Block Diagram (FBD) language aimed to support both a model and a search-based approach. Many benefits emerge from this tool, including the ability to automatically generate test suites from an FBD program in order to comply to quality requirements such as component testing and specific coverage measurements. Search-based testing methods are used to generate test data based on executable code rather than the FBD program, alleviating any problems that may arise from the ambiguities that occur while creating FBD programs. Test cases generated by both approaches are executed and used as a way of cross validation. In the current work, we describe the architecture of the tool, its workflow process, and a case study in which the tool has been applied in a real industrial setting to test a train control management system.

My contribution: I was the second author of this paper. My contributions were mainly the sections that relate to search-based software testing.

Abstract: In this paper, we discuss the application of search-based software testing techniques for unit level testing of a real-world telecommunication middleware at Ericsson. Input data for the system under test consists of nested data structures, and includes non-trivial variables such as uninitialized pointers. Our current implementation analyzes the existing test cases to discover how to handle pointers, set global system parameters, and any other setup code that needs to run before the actual test case. Hill climbing (HC) and (1+1) evolutionary algorithm (EA) metaheuristic search algorithms are used to generate input data for branch coverage. We compare HC, (1+1)EA, and random search as a baseline of performance with respect to effectiveness, measured as branch coverage, and efficiency, measured as number of executions needed. Difficulties arising from the specialized execution environment and the adaptations for handling these problems are also discussed.

My contribution: I was the main author of the paper.

2.3 Research Methodology

In this work we mainly use an applied and experimental approach. Case studies are used as the driving element of the research. In each case study, we model the real-world phenomena such that it can be approached as an optimization problem. Then we experiment with various optimization algorithms and compare the results with the state-of-practice approaches at the particular industry. When a direct comparison with an existing value is not applicable, we compare the optimization approach with simpler algorithms to see how much relative improvement is actually achieved.

The industrial case studies are also used for validation via prototype implementations. In the case studies, we focus on industrial examples of realistic sizes, in order to validate that our work can be extended to industrial practice. Through these case studies we tackle our research questions (Section 2.1) in an industrial context.
Chapter 3

Related Work

In this chapter we survey the related work in maintenance schedule optimization and search based software testing.

3.1 Maintenance Schedule Optimization

Industrial breakdowns can have effects ranging from high costs due to production losses to catastrophic consequences, including physical injury and loss of life. Therefore, maintenance strategies usually include preventive maintenance with the goal of avoiding breakdowns. However, maintenance is costly, and much of it is unnecessary and avoidable. According to Wireman [4], as much as 1/3 of the maintenance costs are due to bad planning, overtime costs, and limited or misuse of preventive maintenance, and are therefore unnecessary. Maggard and Rhyne [5] state that maintenance expenses are usually in the range of 15–40% of the total production cost on a yearly basis, while Coetzee [6] and Bevilacqua and Braglia [7] estimate these expenses to be 15–50% and 15–70%, respectively.

It is therefore reasonable to define good maintenance as when corrective maintenance is kept low while as few preventive maintenance actions as possible are done [8]. One strategy for lowering maintenance costs is condition based maintenance (CBM), in which real-time condition data is used to prioritize and optimize maintenance resources. A CBM support system will ideally monitor the maintained system continuously, allowing a preventive maintenance activity to be performed just before a potential failure becomes critical. Predictive
maintenance adds dynamic lifetime estimates and/or deterioration models to predict the future wear of components. Ideally, CBM in this form will allow maintenance personnel to do only the right things while minimizing spare parts cost, system downtime and time spent on maintenance.

However, to keep maintenance costs and downtime effects on production under control, maintenance still needs to be scheduled and planned in advance. An important challenge is to take uncertain and changing condition data into account in dynamically optimizing the maintenance plan, ideally decreasing the number of unplanned interruptions and allowing the maintenance plan to be adapted to customer requirements.

Maintenance plan optimization has been an active research area since the seminal work of Barlow and Hunter in the 60s [9]. Survey papers by Dekker [10], Budai et al. [11], and Nicolai and Dekker [12] give good overviews of the topic. Dekker and Scarf discuss the state of the art in applications of maintenance optimization models [13]. More generic mathematical maintenance models are also reviewed by Scarf [14].

Yamayee et al. [15] uses dynamic programming to optimize maintenance planning of electric generating equipment, with respect to reliability, production demand, and maintenance cost. However, overall availability as a fraction of the total time is not considered, and the crew and resource model used does not consider the downtime due to day and week rests.

Other approaches to maintenance optimization include Monte Carlo simulations combined with genetic algorithms [16]. In a related approach, Tan and Kramer [17] consider pre-planned maintenance opportunities, which is similar to our approach in paper A (Chapter 5). However, their approach is non-deterministic in contrast to our optimization method.

Wildeman et al. [18] consider maintenance planning for a more restrictive system where certain properties of the cost function must hold, and where potential gain of co-allocating maintenance is constant for all activities. In our model, cost is a function of component costs and indirect costs, resulting from unavailability of the gas turbine due to maintenance (discussed in Chapter 5). This makes our model more expressive, and thus unsolvable using the polynomial solution approach of Wilderman et al. [18].

In multi-unit maintenance models, the system under consideration consists of several units with identical or individual characteristics regarding failure, costs, setup activities, etc. An overview of multi-unit maintenance models is given by Cho and Parlar [19].

Rail vehicle maintenance includes the additional complexity of moving equipment, and research in rail vehicle maintenance therefore often includes
the associated routing problems. An exception is present in the work by Hani et al. [20, 21] that focus on the detailed planning of work performed in the train maintenance facilities only. Cordeau et al. [22] give a survey of models for optimization of train routing and scheduling. In [23, 24], the problem of routing vehicles to the workshop with minimal maintenance costs is solved with the additional sub-problem of grouping maintenance activities such that the number of maintenance occasions is minimized. The problem of determining optimal vehicle routes is NP-hard in general [25], which is why a heuristic method is employed. A related problem has been studied by Anderegg et al. [26], who propose a heuristic routing approach usable in a long-term perspective. However, packaging of maintenance is not considered. Maróti and Kroon [27, 28] also consider the operational maintenance routing problem without considering maintenance packaging. In [27], a multi-commodity flow model is proposed to solve the problem. In [28], an integer programming formulation is presented, and a shortest path heuristic is proposed to solve the problem for a planning horizon of 1–3 days.

3.2 Search-based Software Testing

In their seminal work in 1976, Miller and Spooner published an approach for generating test data [29], which is now known as search-based software testing (SBST). In 1990, Korel [30] extended Miller and Spooner’s approach and devised the branch distance. Later on, the SBST research field started to gain momentum and lead to work that apply various metaheuristic algorithms [31] on testing problems. SBST has received much interest in recent years and a number of survey papers covering the field have been published [32, 33, 34, 35, 36].

An important subtopic in SBST is how the program state is handled, which is also relevant for industrial control applications. Baresel et al. [37] handles the state of a single function by considering a test case to be a sequence of calls, rather than a single call, to the function under test. The fitness value for such a sequence of function calls, is defined as the minimum fitness of individual calls. Tonella [38] devises a similar approach, but for classes in object-oriented programs. It may not be easy to determine the optimal sequence length, such that the input sequence is long enough to reach the correct state, and short enough to avoid wasting resources. Fraser and Arcuri [39] suggest techniques to control the sequence length at run time, instead of trying to set it precisely from the start.

Another technique used to tackle the state problem is the chaining approach,
which was developed by Ferguson and Korel [40]. The chaining approach is based on analyzing the source code and finding the nodes in the control flow graph with internal variables that may need to run to put the program in the correct state. McMinn and Holcombe [41] use the chaining approach as a fail-over mechanism when the primary evolutionary algorithm can not reach the target branch.

Another major challenge in SBST occurs when inputs to a program are complex dynamic data structures or pointers. According to Baars et al. [42], problematic language constructs —such as variable length argument functions, pointers, and complex data types (variable size arrays, recursive types; lists, trees, graphs, etc.)— remain largely unsupported in SBST. Handling complex input data structures is a major challenge for using SBST in an industrial context [43].

Alshraideh and Botacci [44] use a genetic algorithm to generate test data for covering branches with string predicates. Lakhotia et al. [45] combine search-based testing and symbolic execution [46] to generate test data for pointer inputs. They use an alternating variable hill climbing method with a set of constraint solving rules. The problem of path explosion is solved by using a lazy initialization approach for dynamic data structures.

Symbolic execution is also the basis for concolic testing [47, 48]. Concolic testing uses symbolic execution to solve constraints by combining concrete execution with symbolic generation of path conditions. A path condition is constructed in symbolic execution which is a system of constraints in terms of the input variables describing when the path will be executed. The path condition can become too difficult to solve if it contains floating point arithmetic or non-linear constraints. Inkumsah and Xie [49] combine SBST and concolic testing in a framework for structural coverage of Java classes. Hybrid approaches that combine concolic testing and SBST is an ongoing research area [50, 51].
Chapter 4

Conclusions

In this thesis we discussed applications of optimization methods for maintenance scheduling and software testing problems, through multiple industrial case studies. Even though the underlying ideas of optimization are generic and well-known, we needed to adapt the methods for the specific industrial context, in each case study. Such adaptations ranged from being straightforward and easy, to speculative and time consuming to implement. Therefore, a significant portion of our work focused on prototype implementations to show that the optimization approach is feasible. These prototypes also facilitated the experimentation phase in each case study, where we were able to achieve significant and measurable improvements compared to the state of the practice in the given domain and context.

The first problem domain that we focused on was maintenance scheduling. Reducing the total downtime spent on maintenance activities can lead to significant monetary gains on heavy industrial machinery. Two case studies were conducted: firstly on the gas turbines from Siemens Industrial Turbomachinery, and another on a train fleet maintained by EuroMaint. Even though at a higher abstraction level the case studies are similar, both had certain idiosyncrasies and specific requirements not applicable to the other. In both studies we were able to formulate an optimization problem that capture the specific details and achieve significant gains as a result.

The second problem domain that we tackled was software testing, or more specifically test input generation for structural coverage. Again we conducted two industrial case studies with very different requirements at the implementation level. First case study was on train control system of Bombardier Transporta-
tion, while the second study was on a particular telecommunication middleware at Ericsson. In both case studies we automatically parsed and used semantic information from secondary software artifacts (i.e., other than the code being tested). Utilizing such semantic information was crucial in order to produce test cases that are not only achieving code coverage, but are also meaningful when considering the semantics of the system under test. Once again, in both case studies we constructed prototype implementations to demonstrate applicability, and to quantify the performance of our approach on particular industrial systems.

In two different domains, maintenance scheduling and software testing, we showed that the optimization methods can be applied with good results, with a varying degree of improvement on the existing systems. A common theme was that in order to achieve better results that are relevant to the business logic of the company, we needed to incorporate enough context dependent information. The prototype implementations in each case study shows that it is indeed feasible to use optimization methods to achieve significant improvements that are applicable to the phenomena of interest.

4.1 Future Work

In our work we regarded the maintenance scheduling as a deterministic problem. We primarily use predetermined safe boundaries for maintenance deadlines, which is typical in industrial practice. Whenever new deadlines are available (e.g., as a result of a system inspection by human experts), we calculate new schedules. Another approach would be to consider the maintenance deadlines as probabilistic values from the very start. A major challenge is to get realistic probability distributions, as it would be naive to expect domain experts to come up with probability distributions for each maintenance activity. However, with the rise of data awareness and the urge to collect as much operational data as possible, we are seeing opportunities of accessing huge streams of industrial data, albeit being low-level and unfiltered. On the other hand, probabilistic programming paradigms have emerged and matured to a level that is able to cope with big data. Hence, it would be very interesting to approach the scheduling problem with a total probabilistic mindset, as a future endeavor.

In the SBST community there is rising interest in combining search-based methods with various static analysis methods. Certain software structures are relatively easy to manage by the SBST approach and quite difficult when using, let’s say, constraint solvers – while the opposite is true for other software
structures. Combining these two approaches makes a lot of sense, because non-trivial software usually holds both types of structures, and allows neither dynamic nor static approaches to be the silver bullet. However, it is still a challenge to combine multiple approaches in an organic way, such that the results are consistently better than spending half of the resources (typically computation time) on one method, and the rest on the other. In future, we plan to further work on hybrid approaches for software testing problems.


Bibliography


